فهرست



آموزش سریع میکروکنترلر AVR

مولف: رضا سپاس یار (info@avr.ir)

www.avr.ir

www.pnkavosh.com

عنوان مطالب

فیوز بیت ها، منابع کلاک و Reset

- **فیوز بیت ها**

فیوز بیت ها قسمتی از حافظه ی میکروکنترلر AVR هستند که امکاناتی را در اختیار کاربر قرار می دهند و با Erase شدن میکرو مقدار آن ها تغییر نمی کند. یک به معنی غیر فعال بودن و صفر فعال بودن هر بیت می باشد.

قطعه ی Mega16 دارای ۲ بایت فیوز بیت طبق جدول زیر می باشد:

شماره بیت	High Byte	عملکرد	پیش فرض
۰	BOOTRST	انتخاب بردار Reset بخش Boot	۱
۱	BOOTSZ0	انتخاب اندازه ی Bootloader	۰
۲	BOOTSZ1		۰
۳	EESAVE	حفاظت از EEPROM در زمان Erase	۱
٤	CKOPT	انتخاب عملکرد کلاک	۱
۵	SPIEN	فعال ساز پروگرام شدن از طریق SPI	۰
٦	JTAGEN	فعال ساز پورت JTAG	۰
۷	OCDEN	فعال ساز اشکال زدایی از طریق JTAG	۱

شماره بیت	Low Byte	عملکرد	پیش فرض
۰	CKSEL0	انتخاب منبع کلاک	۱
۱	CKSEL1		۰
۲	CKSEL2		۰
۳	CKSEL3		۰
٤	SUT0	انتخاب زمان Startup	۰
٥	SUT1		۱
٦	BODEN	فعال ساز آشکار ساز Brown-out	۱
۷	BODLEVEL	تنظیم سطح ولتاژ Brown-out	۱

BOOTRST: انتخاب بردار ری ست BOOT که در حالت پیش فرض برنامه ریزی نشده است و آدرس بردار ری ست 0000 است و در صورت برنامه ریزی آدرس بردار Reset طبق جدول زیر تعیین می شود. (بر اساس BOOTSZ[1:0])

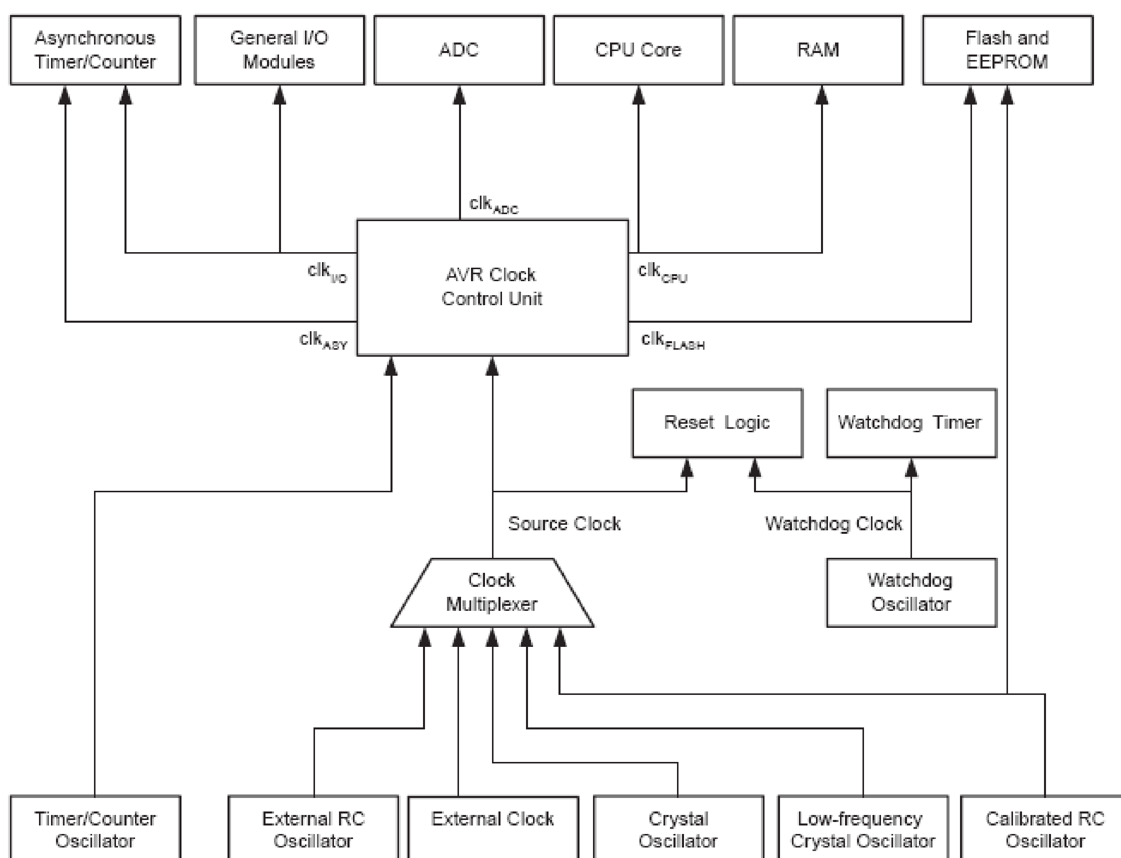
BOOTSZ1	BOOTSZ0	اندازه ی Boot	Pages	آدرس بردار Reset
۱	۱	۱۲۸ Word	۲	$1F80
۱	۰	۲٥٦ Word	٤	$F00
۰	۱	٥۱۲ Word	۸	$E00
۰	۰	۱۰۲٤ Word	۱٦	$C00

BODEN: این بیت فعال ساز Brown-out Detector بوده و در صورت پروگرام شدن مطابق وضعیت جدول زیر سطح ولتاژ Brown-out تعیین می شود.

BODEN	BODLEVEL	سطح ولتاژ Brown-out
۱	۱	غیر فعال
۱	۰	غیر فعال
۰	۱	Vcc=2.7v
۰	۰	Vcc=4.0v

- **منابع کلاک**

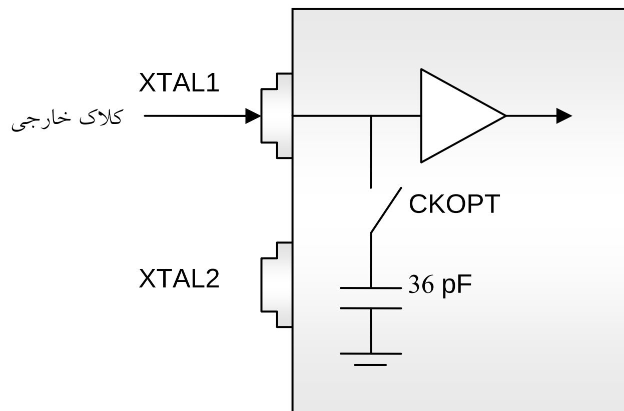
همانطور که در دیاگرام زیر دیده می شود، این منابع شامل: اسیلاتور RC کالیبره شده، اسیلاتور کریستالی فرکانس پایین، اسیلاتور کریستالی، کلاک خارجی، اسیلاتور RC خارجی و اسیلاتور تایمر/کانتر می باشند.



انتخاب منبع کلاک بوسیله ی فیوزبیت های CKSEL بوده و مطابق جدول زیر می باشد. مقدار پیش فرض بیت های CKSEL، یک بوده و در نتیجه منبع پیش فرض، اسیلاتور RC داخلی می باشد.

Device Clocking Option	CKSEL3..0
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

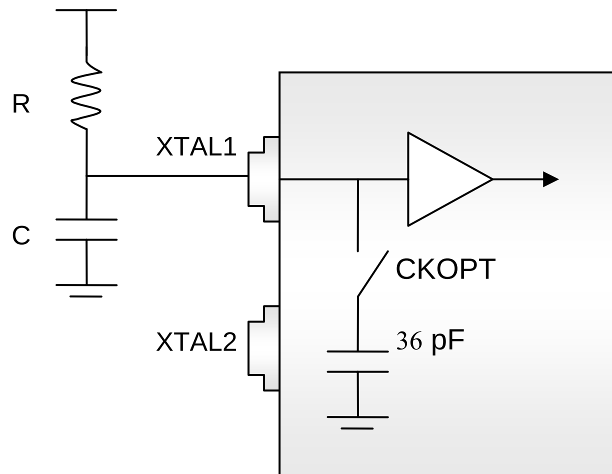
کلاک خارجی: برای راه اندازی وسیله بوسیله ی منبع کلاک خارجی باید مطابق شکل زیر یک پالس به پین XTAL1 اعمال شود. برای قرار گرفتن در این وضعیت باید تمام بیت های CKSEL پروگرام شده (صفر شوند) و کاربر می تواند با پروگرام کردن فیوزبیت CKOPT یک خازن داخلی به ظرفیت ۳٦ پیکوفاراد را بین ورودی و زمین قرار دهد.



اسیلاتور RC کالیبره شده ی داخلی: این منبع در فرکانس های ۱، ۲، ٤ و ۸ مگاهرتز موجود می باشد و مقدار آن در دمای ۲۵ درجه و ولتاژ ٥ ولت کالیبره شده است که در این وضعیت ممکن است تا ۳ درصد در کلاک ایجاد شده وجود داشته باشد. فرکانس نوسان بوسیله ی فیوزبیت های CKSEL تعیین شده و مطابق جدول زیر می باشد. در این وضعیت CKOPT نباید پروگرام شود.

CKSEL3..0	Nominal Frequency (MHz)
0001(1)	1.0
0010	2.0
0011	4.0
0100	8.0

اسیلاتور RC خارجی: در کاربردهایی که دقت کلاک اهمیت زیادی ندارد می توان از این منبع استفاده کرد. پیکربندی مطابق شکل زیر بوده و فرکانس نوسان از رابطه ی $f = \frac{۱}{۳RC}$ بدست می آید. حداقل مقدار C برابر ۲۲ پیکوفاراد بوده و در صورتی که CKOPT پروگرام شود می توان مقدار ۳٦ پیکوفاراد را نیز لحاظ نمود.

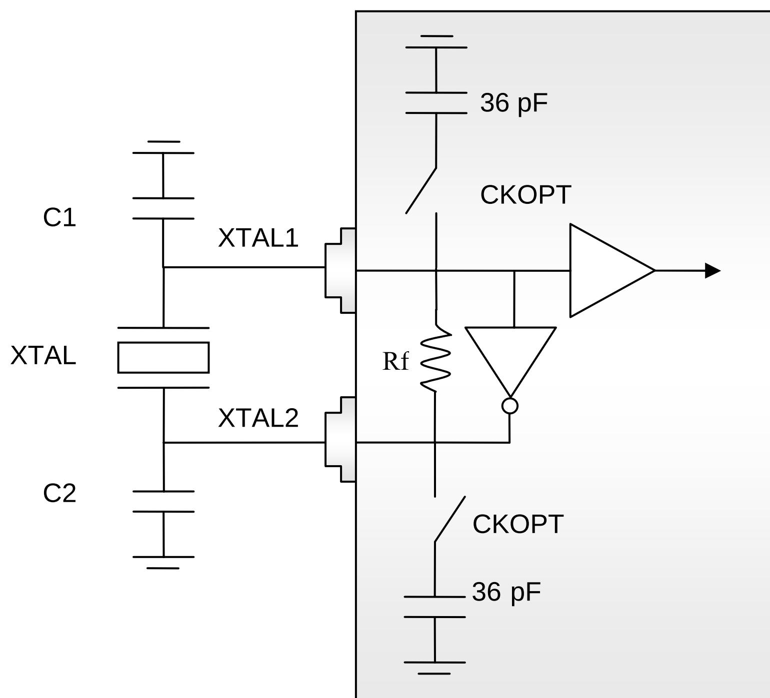


این منبع نوسان می تواند در چهار Mode کاری عمل کند که هر کدام برای یک بازه ی فرکانسی بهینه شده است و بوسیله ی فیوزبیت های CKSEL مطابق جدول زیر انتخاب می شود.

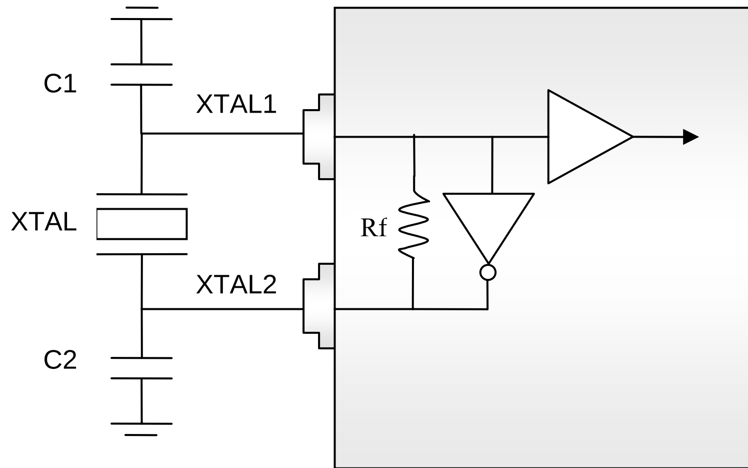
CKSEL3..0	Frequency Range (MHz)
0101	≤ 0.9
0110	0.9 - 3.0
0111	3.0 - 8.0
1000	8.0 - 12.0

اسیلاتور کریستالی فرکانس پایین: این منبع کلاک می تواند کریستال های فرکانس پایین مثل کریستال ساعت با فرکانس ۳۲۷٦۸ هرتز باشد. با دادن مقدار ۱۰۰۱ به فیوزبیت های CKSEL منبع کلاک کریستال خارجی فرکانس

پایین انتخاب شده و در این وضعیت پیکربندی مطابق شکل زیر می باشد. در صورت پروگرام نمودن CKOPT می توان از خازن خارجی صرفنظر نمود.



کریستال کوارتز یا رزوناتور سرامیکی: پبن های XTAL1 و XTAL2 به ترتیب ورودی و خروجی یک تقویت کننده ی وارونگر هستند که می توانند به عنوان یک اسیلاتور On-chip مطابق شکل زیر پیکربندی شوند.



- به جای کریستال کوارتز می توان از رزوناتور سرامیکی استفاده نمـود کـه از دوام بیشـتری در مقابـل ضـربه بـر خوردار است و زمان Startup کمتری نیز دارد و البته نسبت به کریستال کوارتز دقـت کمتـری داشـته و پایـداری دمایی آن نیز کمتر است.

- در این وضعیت خازن های ٣٦ پیکو فاراد حذف شده و عملکرد فیوزبیت CKOPT نیز تغییر مـی کنـد. بـدین ترتیب که با پروگرام شدن این بیت دامنه ی خروجی تقویت کننده ی وارونگر افـزایش یافتـه و مـی تـوان از پـین XTAL2 به عنوان کلاک برای یک وسیله ی دیگر استفاده نمود. همچنین با فعال کردن CKOPT در محیط های نویزی عملکرد اسیلاتور بهبود می یابد.

- چنانچه از رزوناتور استفاده می شود برای فرکانس های بالاتر از ٨ مگاهرتز باید CKOPT پروگرام شود.

اسیلاتور می تواند در سه وضعیت متفاوت نوسان کند که هرکدام برای یک محدوده ی فرکانسی بهینه شده است و آن را می توان با فیوز بیت های CKSEL مطابق جدول زیر انتخاب نمود.

CKOPT	CKSEL3..1	Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF)
1	101[1]	0.4 - 0.9	–
1	110	0.9 - 3.0	12 - 22
1	111	3.0 - 8.0	12 - 22
0	101, 110, 111	1.0 ≤	12 - 22

Note: 1. This option should not be used with crystals, only with ceramic resonators.

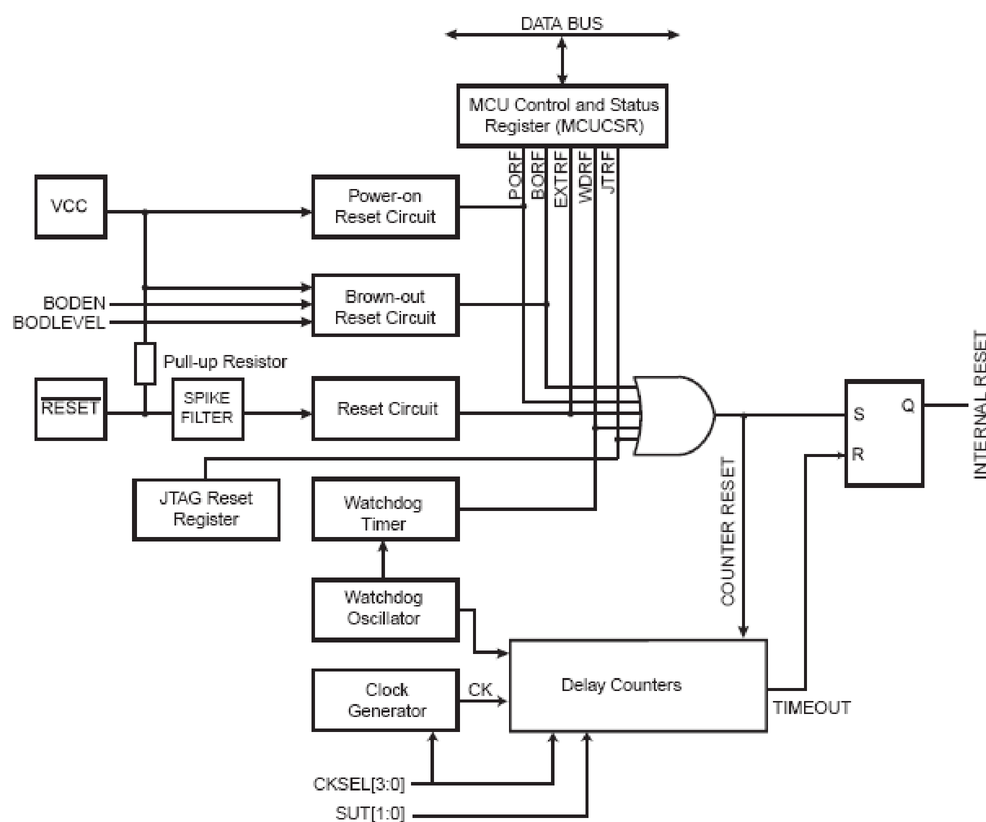
با هر یک از منابع کلاک انتخاب شده بوسیله ی فیوزبیت های CKSEL، دو بیت به نام های SUT[1:0] نیز وجود دارد که از طریق آن می توان حداکثر زمان Start-up منبع کلاک را به میکرو اعلام نمود. مقدار این بیت ها به طور پیش فرض ماکزیمم زمان Start-up را در نظر می گیرد و در صورتی کـه نیـاز اسـت مقـدار آن را تغییـر دهید مطابق جداول مربوطه در فصل System Clock and Clock Options در Datatsheet عمل کنید.

- **منابع Reset**

با Reset شدن میکروکنترلر، تمام رجیسترهای I/O به مقدار اولیه شان تغییر می کنند و CPU شروع به اجرای دستورالعمل ها از بردار Reset خواهد کرد. در قطعه ی Mega16 ۵ منبع Reset وجود دارد که عبارتند از:

1. Power-on Reset
2. External Reset
3. Brown-out Reset
4. Watchdog Reset
5. JTAG AVR Reset

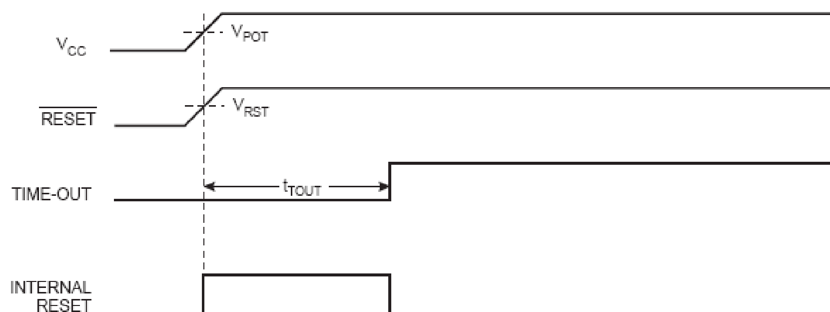
منطق Reset مطابق دیاگرام زیر می باشد:



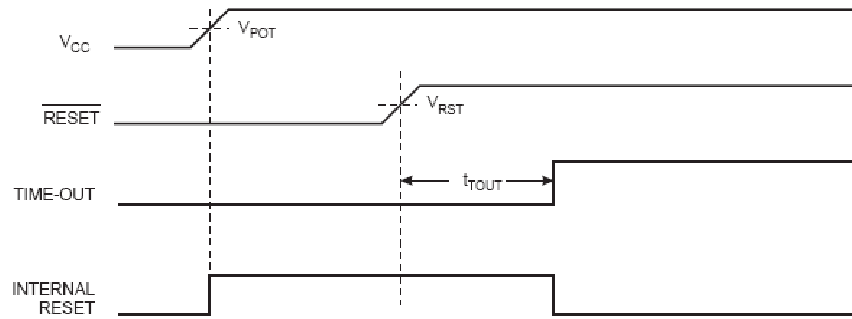
مشخصات هر یک از منابع Reset را در جدول زیر مشاهده می کنید:

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{POT}	Power-on Reset Threshold Voltage (rising)			1.4	2.3	V
	Power-on Reset Threshold Voltage (falling)(1)			1.3	2.3	V
V_{RST}	$\overline{RESET}$ Pin Threshold Voltage		0.1 V_{CC}		0.9V_{CC}	V
t_{RST}	Minimum pulse width on $\overline{RESET}$ Pin				1.5	µs
V_{BOT}	Brown-out Reset Threshold Voltage(2)	BODLEVEL = 1	2.5	2.7	3.2	V
		BODLEVEL = 0	3.7	4.0	4.2	
t_{BOD}	Minimum low voltage period for Brown-out Detection	BODLEVEL = 1		2		µs
		BODLEVEL = 0		2		µs
V_{HYST}	Brown-out Detector hysteresis			50		mV

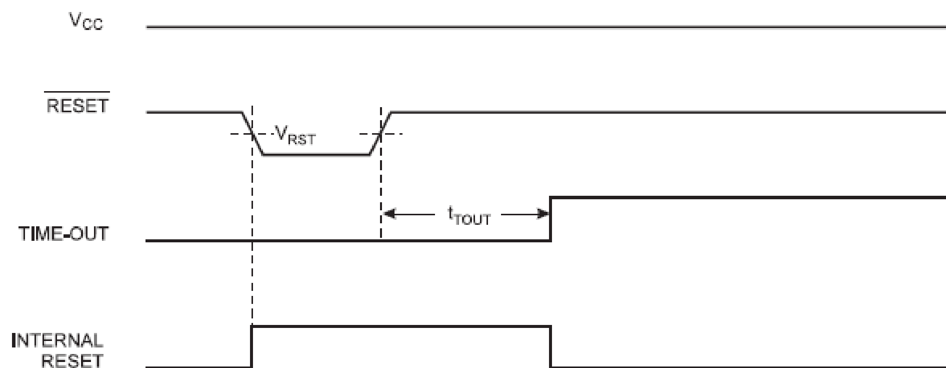
۱. **Power-on Reset**: زمانی فعال خواهد که ولتاژ Vcc کمتر از حد تعیین شده باشد. این منبع تضمین می کند که وسیله در زمان راه اندازی Reset می شود. با رسیدن ولتاژ به حد آستانه، شمارنده ی تاخیر راه اندازی شده که تعیین می کند چه مدت وسیله در وضعیت Reset بماند. دیاگرام زمانی زیر شرایطی را نشان می دهد که پین $\overline{RESET}$ به Vcc وصل شده است. (و یا آزاد باشد چون این پین از داخل Pull-up شده است.)



و نمودار زیر شرایطی است که سطح منطقی پین Reset تابع Vcc نمی باشد:



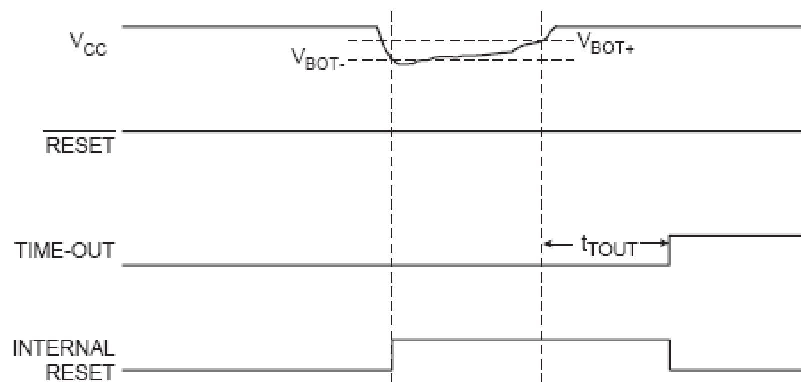
۲. **External Reset**: این Reset بوسیله ی یک پالس با سطح صفر منطقی روی پین $\overline{\text{RESET}}$ ایجاد شده و حداقل عرض آن ۱.۵ میکرو ثانیه می باشد. با رسیدن ولتاژ این پین به مقدار آستانه در لبه بالا رونـده، شـمارنده ی تاخیر شروع به کار کرده و پس از اتمام زمان Time-out میکروکنترلر کار خود را شروع خواهد کرد.



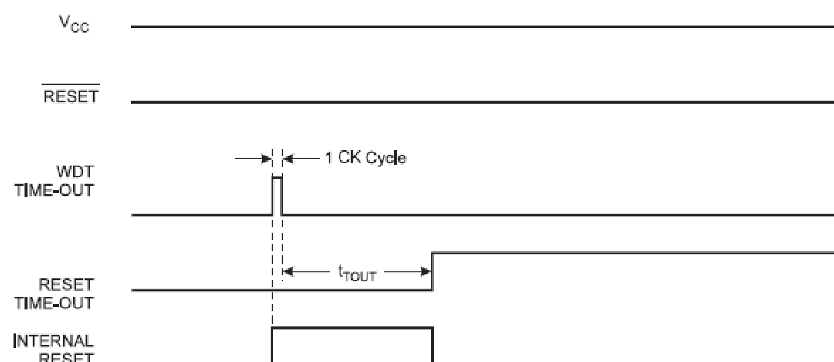
۳. **Brown-out Detection**: قطعـه ی ATmega16 دارای یـک مـدار Brown-out Detection داخلی بوده که پیوسته مقدار ولتاژ Vcc را با یک مقدار ثابت مقایسه می کند. این مقدار ثابت برابر ۲.۷ ولت بـوده و در صورتی که فیوزبیت BODLEVEL پروگرام شود به ٤.٠ ولت افزایش می یابد. با کمتر شدن ولتـاژ تغذیـه از این مقدار ثابت میکروکنترلر وارد حالت Reset شده و با عادی شدن ولتاژ، پـس از اتمـام تـاخیر بـه وضـعیت عادی باز می گردد. برای حفاظت در برابر Spike مقدار آستانه دارای پسـماند بـوده و در نتیجـه دارای دو مقـدار

مثبت و منفی می باشد که با توجه به مقادیر موجود در جدول از رابطه ی $V_{BOT+} = V_{BOT} + V_{HYST} / ۲$ و $V_{BOT-} = V_{BOT} - V_{HYST} / ۲$ بدست می آید.

- مدار Brown-out Detection در حالت عادی غیر فعال بوده و برای راه اندازی آن باید فیوزبیت BODEN پروگرام (صفر) شود.



٤. **Watchdog Reset**: با اتمام زمان تایمر Watchdog، این تایمر یک پالس به عرض یک سیکل ایجاد خواهد کرد. در لبه ی پایین رونده ی این پالس، تایمر تاخیر شروع به شمارش زمان تاخیر کرده و پس ازاتمام آن میکروکنترلر کار عادی خود را ادامه خواهد داد.



MCU Control and Status Register

MCUCSR	7	6	5	4	3	2	1	0
نام بیت	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF

این رجیستر محتوی اطلاعاتی است که نشان می دهد کدامیک از منابع Reset باعث راه اندازی مجدد CPU شده است. نرم افزار پس از خواندن هر بیت باید با نوشتن صفر بر روی آن، پرچم را پـاک کنـد تـا در صـورت Reset مجدد، وقوع آن قابل تشخیص باشد.

Bit 0 – PORF: Power-on Reset Flag
Bit 1 – EXTRF: External Reset Flag
Bit 2 – BORF: Brown-out Reset Flag
Bit 3 – WDRF: Watchdog Reset Flag
Bit 4 – JTRF: JTAG Reset Flag

آشنایی با زبان C

- هر برنامه ی C حداقل یک تابع main() دارد. (این اولین تابع اجرایی است.)
- یک برنامه ی میکروکنترلری در ساده ترین حالت خود به شکل زیر است:

تعاریف کلی

الگوی توابع

```
void main()
{
        while(1){
                .
                .
                .
        }
}
```

توابع تعریف شده

- برنامه ی ساده ی زیر رشته ی Hello World را به خروجی استاندارد ارسال می کند.

```
#include<stdio.h>

void main()
{
    printf("Hello World!");
    while(1);
}
```

- خط اول از رهنمود های پیش پردازنده است.
- while(1) ایجاد یک حلقه ی نامتناهی می کند.
- در انتهای هر عبارت یک سمی کالن می آید.
- Brace ابتدا و انتهای یک تابع و همچنین یک بلوک را مشخص می کند.
- از " " برای مشخص کردن ابتدا و انتهای یک رشته ی متنی استفاده می شود.
- از // یا /* ... */ برای نوشتن توضیحات استفاده می شود.
- شناسه ها اسامی متغیرها، ثوابت و یا توابع هستند.
- شناسه ها نمی تواند از کلمات رزرو شده باشند و همچنین نمی توانند با یک کاراکتر عددی شروع شود و طول آن ها باید کمتر از ۳۱ کاراکتر باشد.
- C یک زبان Case Sensitive است و بین حروف کوچک و بزرگ تفاوت قائل می شود.
- کلمات رزرو شده حتما باید با حرف کوچک استفاده شوند. (مثل ...,if, char, while)

متغیر ها و ثوابت

- تعریف متغیر یعنی انتخاب نام مستعار برای مکانی از حافظه
- فرم تعریف متغیر: نام متغیر ; نوع متغیر مثال: char a ;

انواع داده ها:

Type	Size (Bits)	Range
bit	1	0 , 1
char	8	-128 to 127
unsigned char	8	0 to 255
signed char	8	-128 to 127
int	16	-32768 to 32767
short int	16	-32768 to 32767
unsigned int	16	0 to 65535
signed int	16	-32768 to 32767
long int	32	-2147483648 to 2147483647
unsigned long int	32	0 to 4294967295
signed long int	32	-2147483648 to 2147483647
float	32	±1.175e-38 to ±3.402e38
double	32	±1.175e-38 to ±3.402e38

- برای تعیین محل تعریف متغیر از عملگر @ استفاده می کنیم. مثال: int b@0xA3;

- برای تعریف متغیر در حافظه ی EEPROM از کلمه ی کلیدی eeprom قبل از نام متغیر استفاده می کنیم.

مثال: eeprom int code;

- می توان در زمان تعریف به متغیر مقدار اولیه داد. مثال: char s = 'a';
- تعریف ثابت با کلمه ی کلیدی const یا flash انجام می شود. مثال: const float pi = 3.14 ;
- با رهنمود #define می توان ماکرو تعریف نمود، در این حالت مقداری که برای ثابت تعریف می شود نوع داده را تعیین کرده و مقادیر تعریف شده، توسط پیش پردازنده با مقدار ثابت جایگزین می شود.

مثال: #define code 100 ;

عملگرهای حسابی و بیتی

عملگر	عملکرد	مثال	نتیجه
*	ضرب	۳*۲	٦
/	تقسیم	۵/۲	۲.۵
+	جمع	۳+٦	۹
-	تفریق	۸-۳	۵
%	باقیمانده	۱۰٪۳	۱
&	AND	0xF0 & 0x0F	0x00
|	OR	0x00 | 0x03	0x03
^	XOR	0x0F ^ 0xFF	0xF0
~	مکمل یک	~(0xF0)	0x0F
>>	شیفت به راست	0xF0 >> 4	0x0F
<<	شیفت به چپ	0x0F << 4	0xF0

عملگرهای یکانی

عملگر	عملکرد	مثال	نتیجه
-	قرینه	-a	a = a×-1
++	افزایش یک واحد	a++	a = a + 1
--	کاهش یک واحد	a--	a = a - 1

عملگرهای مقایسه ای

عملگر	عملکرد	مثال	نتیجه
>	بزرگتر	2>3	False
<	کوچکتر	'm'>'e'	True
>=	بزرگتر یا مساوی	5>=5	True
<=	کوچکتر یا مساوی	2.5<=4	True
==	تساوی	'A' =='B'	False
!=	نامساوی	2!=3	True

عملگرهای منطقی

عملگر	عملکرد	مثال	نتیجه
&&	AND منطقی	(2>3)&&(1!=3)	False
||	OR منطقی	('a'<3)||(10)	True
!	نقیض	!(7>5)	False

عملگرهای انتساب

عملگر	عملکرد	مثال	نتیجه
=	انتساب	a=b	a ⇦ b
*=	ضرب و انتساب	a*=b	a=a*b
/=	تقسیم و انتساب	a/=b	a=a/b
+=	جمع و انتساب	a+= b	a=a+b
-=	تفریق و انتساب	a-=b	a=a-b
a=(value)?x:y	انتساب شرطی		اگر value مقدار صحیح باشد a برابر x و در غیراینصورت برابر y می شود.

آرایه ها

آرایه ها مجموعه ای از متغیرهای همنوع هستند.

- فرم تعریف: ; [تعداد عناصر]اسم متغیر نوع متغیرهای آرایه
- مثال: int a[5];

با مقدار اولیه: int a[5] = {1,2,3,4,5};

- فرم تعریف آرایه های دو بعدی: ;[عداد عناصر ستون][تعداد عناصر سطر]اسم متغیر نوع متغیرها

مثال: int a[2][3]={{1,2,3},{4,5,6}};

- در زبان C اندیس آرایه ها از صفر شروع می شود.

رشته ها

رشته ها آرایه ای کاراکترها هستند.

- مثال: char name[ ] = "Test";

یا: char name[5]={'T','e','s','t','\0'};

- رشته ها همواره به یک کاراکتر Null ختم می شوند.

تصمیم گیری و انتخاب

① دستور goto:

پرش بدون شرط به یک برچسب انجام می شود.

② ساختار if – else:

```
if(شرط)
{
      دستورات۱;
}
else
{
      دستورات ۲;
}
```

- ✓ در صورتی که دستورات یک خطی باشند می توان brace را حذف نمود.
- ✓ استفاده از بلاک else اختیاری است.

مثال:

```
#include<stdio.h>

void main(){
```

```
    int a;

    printf("Enter a Number: ");
    scanf("%d",&a);

    if(a= =0)
        printf("You've entered zero\n");
    else if(a>0)
        printf("You've entered a positive number\n");
    else
        printf("You've entered a negative number\n");
}
```

③ ساختار **Switch - Case**:

```
switch (مقدار)
{
    case   مقدار۱:
    دستورات۱;
    break;
    case   مقدار۲:
    دستورات۲;
    break;
```

```
    .
    .
    default:
    n دستورات;
}
```

مثال:

```
#include<stdio.h>

void main(){

    int a;
    printf("Enter Month of your birth: ");
    scanf("%d",&a);

    switch(a){

     case 1:
     case 2:
     case 3: printf("You've born is spring\n");
     break;

     case 4:
     case 5:
     case 6: printf("You've born is summer\n");
     break;

     case 7:
     case 8:
     case 9: printf("You've born is autumn\n");
```

```
     break;

     case 10:
     case 11:
     case 12: printf("You've born is winter\n");
     break;

     default: printf("Error! Enter a number between 1-
    12\n");

    }

}
```

حلقه های تکرار

① ساختار While:

```
while(شرط)
{
    دستورات;
}
```

مثال:

```
#include<stdio.h>

void main(){
```

```
    char a;

    printf("Enter E to exit\n");
    while(a != 'E') a=getchar();

}
```

② ساختار **Do/While**:

```
do{
    دستورات;
} while(شرط)
```

✓ در ساختار Do/While بر خلاف While شرط در انتهای حلقه آزمایش می شود، بنابراین دستورات داخل حلقه، حداقل یکبار اجرا می شوند.

③ حلقه های **For**:

```
for(مقدار ابتدای حلقه ;  شرط پایان ;  گام)
    {
    دستورات;
    }
```

مثال:

```
#include<stdio.h>

void main(){

    int a,i;
    long int fact=1;

    printf("Enter a Number: ");
    scanf("%d",&a);

    if(a<0)
     printf("Error! You must Enter a positive number\n");
    else if(a==0)
     printf("Factorial of 1 is 1\n");
    else{
     for(i=1;i<=a;i++)
          fact*=i;
     printf("Factorial of %d is %d\n",a,fact);
    }
}
```

- ✓ دستور break باعث خروج بدون شرط از هر حلقه ای می شود.
- ✓ دستور continue باعث می شود اجرای ادامه دستورات متوقف شده و حلقه از ابتدا آغاز شود.

توابع

تعریف توابع به صورت زیر می باشد:

```
نوع داده خروجی     نام تابع (آرگومانهای تابع)
{
        متغیرهای محلی
        دستورات تابع
}
```

⇦ توابع داخل یکدیگر قابل تعریف نمی باشند و جدا از هم باید تعریف گردند.

مثال:

```
#include <stdio.h>

long int cube(int x);

void main(){
```

```
      int a;

      printf("Enter a number: ");
      scanf("%d",&a);
      printf("Cube of %d is %d\n",a,cube(a));

}

long int cube(int x){
      return x*x*x;
}
```

مثال:

```
#include<stdio.h>

int _max(int a,int b);

void main(){

    int a,b;

    printf("Enter Two Numbers: ");
    scanf("%d%d",&a,&b);
    printf("Maximum of %d and %d is %d\n",a,b,_max(a,b));
```

```
}


int _max(int a,int b){
     if(a>b)
          return a;
     else
          return b;
}
```

پروژه ۱: فلاشر ساده

```
/***************************************************
Project : LED Flasher
Author  : Reza Sepas Yar
Company : Pishro Noavaran Kavosh
***************************************************/

#include<mega16.h>
#include<delay.h>
#define xtal 1000000

int i;

void main (void)
{

    DDRA = 0xFF;     //.برای درک عملکرد این عبارت به پیوست ۱ مراجعه کنید

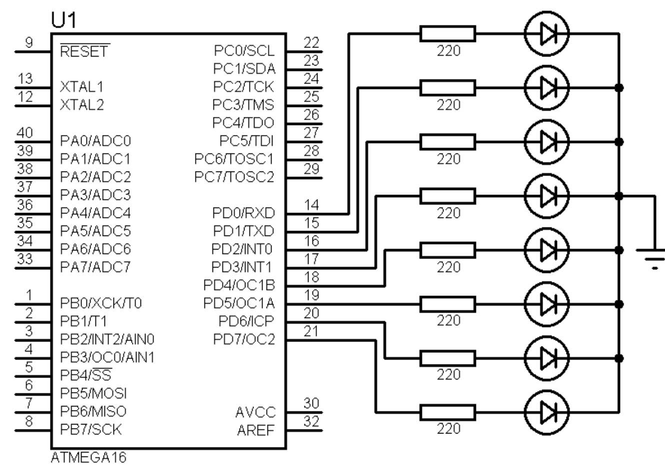
    while(1)
    {

     for(i = 1; i <= 128; i = i*2)
     {
          PORTD = i;
          delay_ms(100);
     }
```

```
     for(i = 64; i > 1; i = i/2)
     {
          PORTD = i;
          delay_ms(100);
     }

    }

}
```



پروژه ۲: کانتر یک رقمی با 7-Segment

```
/*****************************************************
Project : Key Counter
Author  : Reza Sepas Yar
Company : Pishro Noavaran Kavosh
*****************************************************/

#include <mega16.h>
#define xtal 4000000

flash char digits[16]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,
0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71};
unsigned char p_state;
unsigned char key;
unsigned char i;

void main(void)
{
    DDRD = 0xFF;
    PORTD = digits[0];
    DDRC = 0x00;
    PORTC = 0xFF;

    while(1)
    {

     key = PINC & 0b00000001;
```

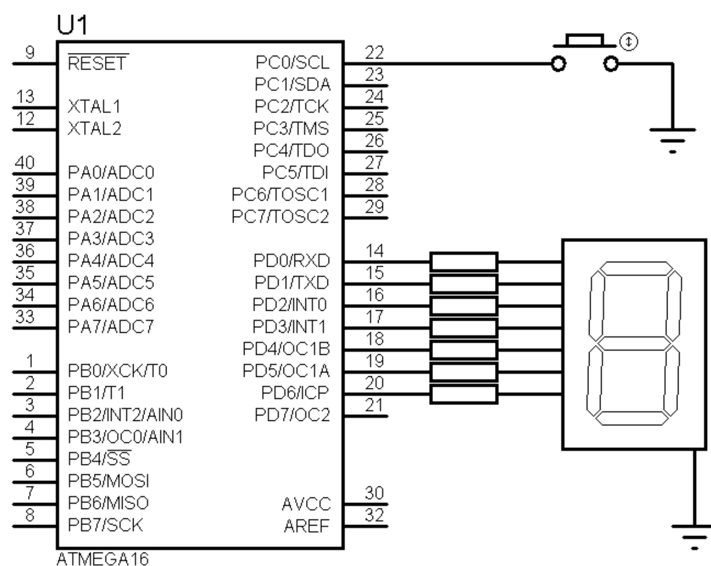
```
 delay_ms(10);


 if(key==0)
 {
         if(key!=p_state)
         {
              if(i==15)
              {
                      i=0;
                  PORTD=digits[i];
            }
           else
                 i++;


          PORTD = digits[i];
            p_state=0;


         };
 }
 else
     p_state=1;
}


}
```



جدول زیر وضعیت سگمنت های 7-Seg کاتد مشترک را نشان می دهد. (برای نوع آند مشترک اعدادرا مکمل کنید.)

کاراکتر	Dp	g	f	e	d	c	b	a	HEX
0	0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5B
3	0	1	0	0	1	1	1	1	0x4F
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7F
9	0	1	1	0	1	1	1	1	0x6F
A	0	1	1	1	0	1	1	1	0x77
b	0	1	1	1	1	1	0	0	0x7C

C	0	0	1	1	1	0	0	1	0x39
d	0	1	0	1	1	1	1	0	0x5E
E	0	1	1	1	1	0	0	1	0x79
F	0	1	1	1	0	0	0	1	0x71

پروژه ۳: نمایشگر کریستال مایع (LCD)

```
/*****************************************************
Project : LCD Interfacing
Author  : Reza Sepas Yar
Company : Pishro Noavaran Kavosh
*****************************************************/

#include <stdio.h>
#include <mega16.h>
#include <delay.h>
#include <lcd.h>
#define xtal 4000000


#asm
   .equ __lcd_port=0x1B ;PORTA
#endasm

void main(void)
{
    char buffer[10];
    unsigned char w;

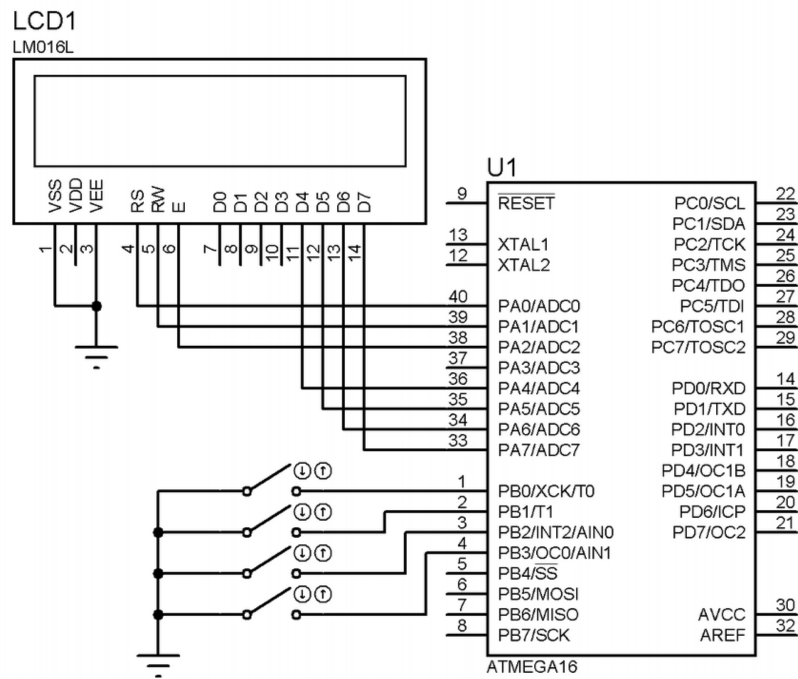
    PORTB=0xFF;
    DDRB=0x00;
```

```
    lcd_init(16);
    lcd_clear();

    while (1){

        w = ~PINB;

        if(w!=0x00)
        {
            lcd_clear();
            lcd_gotoxy(0,0);
            sprintf(buffer,"Number=%d",w);
            lcd_puts(buffer);
            delay_ms(100);
         }
         else
         {
            lcd_clear();
            lcd_putsf("Number=0");
            delay_ms(100);
         }
    }
}
```


LCD1
LM016L
VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/XCK/T0
PB1/T1
PB2/INT2/AIN0
PB3/OC0/AIN1
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP
PD7/OC2
AVCC
AREF
ATMEGA16

پروژه ٤: اسکن صفحه کلید ماتریسی

```
/*****************************************************
Project : Keypad Scan
Author  : Reza Sepas Yar
Company : Pishro Noavaran Kavosh
*****************************************************/

#include <mega16.h>
#include <delay.h>

#define xtal 4000000

unsigned char key, butnum;

flash unsigned char keytbl[16]={0xee, 0xed, 0xeb, 0xe7,
    0xde, 0xdd, 0xdb, 0xd7, 0xbe, 0xbd, 0xbb, 0xb7, 0x7e,
    0x7d, 0x7b, 0x77};

void main(void)
  {
  DDRB = 0xff;
  PORTB = 0xff;

  while(1)
    {
     DDRC = 0x0f;
```

```
    PORTC = 0xf0;
    delay_us(5);
    key = PINC;

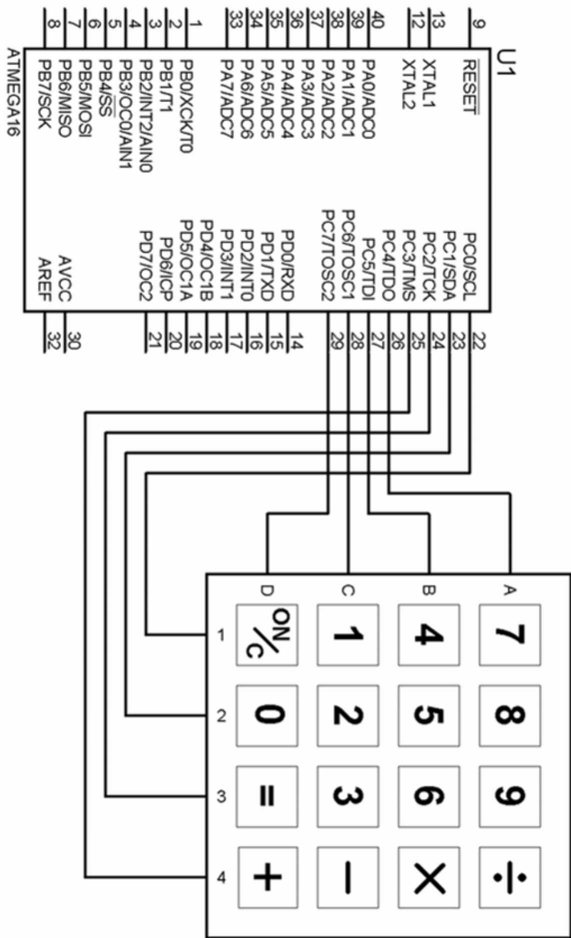
    DDRC = 0xf0;
    PORTC = 0x0f;
    delay_us(5);
    key = key | PINC;

    delay_ms(10);

    if (key != 0xff)
    {
       for (butnum=0; butnum<16; butnum++)
       {
           if (keytbl[butnum]==key)  break;
       }
       if (butnum==16) butnum=0;
            else butnum++;
    }
    else butnum=0;

    PORTB = ~ butnum ;

   }
 }
```


U1
ATMEGA16
9 RESET
13 XTAL1
12 XTAL2
40 PA0/ADC0
39 PA1/ADC1
38 PA2/ADC2
37 PA3/ADC3
36 PA4/ADC4
35 PA5/ADC5
34 PA6/ADC6
33 PA7/ADC7
1 PB0/XCK/T0
2 PB1/T1
3 PB2/INT2/AIN0
4 PB3/OC0/AIN1
5 PB4/SS
6 PB5/MOSI
7 PB6/MISO
8 PB7/SCK
PC0/SCL 22
PC1/SDA 23
PC2/TCK 24
PC3/TMS 25
PC4/TDO 26
PC5/TDI 27
PC6/TOSC1 28
PC7/TOSC2 29
PD0/RXD 14
PD1/TXD 15
PD2/INT0 16
PD3/INT1 17
PD4/OC1B 18
PD5/OC1A 19
PD6/ICP 20
PD7/OC2 21
AVCC 30
AREF 32
A B C D
1 2 3 4
7 8 9 ÷
4 5 6 ×
1 2 3 −
ON/C 0 = +

```
/****************************************************
Project : Dot Matrix Display
Author  : Reza Sepas Yar
Company : Pishro Noavaran Kavosh
****************************************************/

#include <mega16.h>
#include <delay.h>
#define xtal 4000000

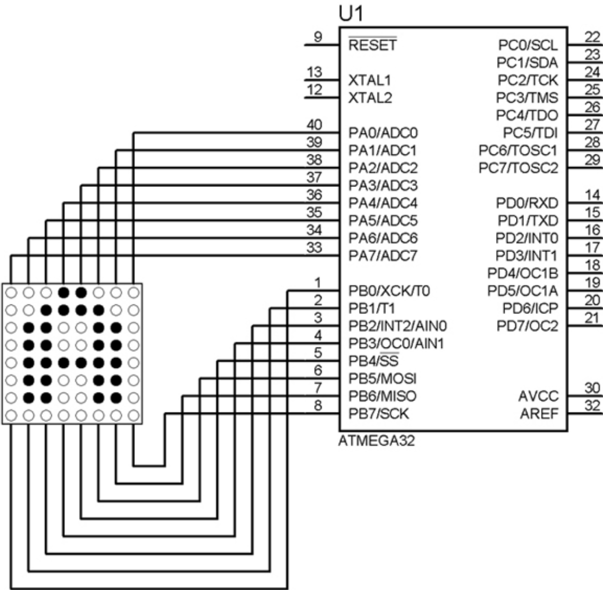
unsigned char k;
flash unsigned char arr[8]={0x18, 0x3C, 0x66, 0x66, 0x7E,
    0x66, 0x66, 0x00};

void main(void)
{

DDRA=0xFF;
DDRB=0xFF;

while (1){
                for(k=0;k<=7;k++){
                        PORTA=arr[k];
                        PORTB=~(1<<k);
                        delay_us(100);
```

```
                    PORTB=0xFF;
          }
     }
}
```


U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/XCK/T0
PB1/T1
PB2/INT2/AIN0
PB3/OC0/AIN1
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP
PD7/OC2
AVCC
AREF
ATMEGA32

وقفه های خارجی

- قطعه ی ATmega16 دارای ۲۱ منبع وقفه می باشد که ۳ مورد از آن ها وقفه ی خارجی می باشند.

شماره	نام در CodeVision	آدرس	منبع وقفه	توضیح
۱		$000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
۲	EXT_INT0	$002	INT0	External Interrupt Request 0
۳	EXT_INT1	$004	INT1	External Interrupt Request 1
٤	TIM2_COMP	$006	TIMER2 COMP	Timer/Counter2 Compare Match
٥	TIM2_OVF	$008	TIMER2 OVF	Timer/Counter2 Overflow
٦	TIM1_CAPT	$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
۷	TIM1_COMPA	$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
۸	TIM1_COMPB	$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
۹	TIM1_OVF	$010	TIMER1 OVF	Timer/Counter1 Overflow
۱۰	TIM0_OVF	$012	TIMER0 OVF	Timer/Counter0 Overflow
۱۱	SPI_STC	$014	SPI, STC	Serial Transfer Complete
۱۲	USART_RXC	$016	USART, RXC	USART, Rx Complete
۱۳	USART_DRE	$018	USART, UDRE	USART Data Register Empty
۱٤	USART_TXC	$01A	USART, TXC	USART, Tx Complete
۱٥	ADC_INT	$01C	ADC	ADC Conversion Complete
۱٦	EE_RDY	$01E	EE_RDY	EEPROM Ready
۱۷	ANA_COMP	$020	ANA_COMP	Analog Comparator
۱۸	TWI	$022	TWI	Two-wire Serial Interface

۱۹	EXT_INT2	$024	INT2	External Interrupt Request 2
۲۰	TIM0_COMP	$026	TIMER0 COMP	Timer/Counter0 Compare Match
۲۱	SPM_READY	$028	SPM_RDY	Store Program Memory Ready

- آدرس های پایین تر دارای اولویت بالاتری می باشند و در صورت درخواست همزمان دو یا چند وقفه ابتدا به اولویت بالاتر پاسخ داده می شود و پس از آن به بقیه ی وقفه ها بر حسب اولویت رسیدگی می شود.

برای فعال کردن هر یک از وقفه ها باید ابتدا بیت فعال ساز عمومی وقفه ها را با دستور اسمبلر SEI یا مقدار دهی رجیستر SREG فعال نمود:

Bit	7	6	5	4	3	2	1	0
SREG	I							

- با رویداد هر وقفه ی خارجی این بیت پاک شده و در نتیجه تمام وقفه های دیگر غیر فعال مـی شـوند در این حالت نرم افزار می تواند با نوشتن یک روی این بیت آن را مجددا فعال کند و باعث ایجاد وقفه هـای تو در تو شود. با بازگشت از ISR این بیت مجددا یک می شود.

برای استفاده از هریک از وقفه های خارجی باید با یک کردن بیت مربوطه در رجیستر GICR آن را فعال نمود:

Bit	7	6	5	4	3	2	1	0
GICR	INT1	INT0	INT2					

- وقفه های خارجی ۱،۰ و ۲ به ترتیب از پین های PD2، PD3 و PB2 تریگر می شوند.

نوع تریگر شدن هریک از وقفه های خارجی ۰ و ۱ بوسیله ی چهاربیت اول رجیستر MCUCR تعیین می شود:

Bit	7	6	5	4	3	2	1	0
MCUCSR					ISC11	ISC10	ICS01	ISC00

حالت های مختلف بیت های MCUCR[3:0]:

ISC01	ISC00	نوع تریگر شدن وقفه ی صفر
0	0	سطح منطقی صفر در پین INT0
0	1	تغییر در سطح منطقی پین INT0
1	0	لبه ی پایین رونده در پین INT0
1	1	لبه ی بالا رونده در پین INT0

ISC11	ISC10	نوع تریگر شدن وقفه ی یک
0	0	سطح منطقی صفر در پین INT1
0	1	تغییر در سطح منطقی پین INT1
1	0	لبه ی پایین رونده در پین INT1
1	1	لبه ی بالا رونده در پین INT1

نوع تریگر شدن وقفه ی ۲ خارجی بوسیله بیت ٦ از رجیستر MCUCSR تعیین می شود:

Bit	7	6	5	4	3	2	1	0
MCUCSR		ISC2						

- وقفه ی۲ خارجی بر خلاف وقفه ی ۰ و ۱ تنها در دو حالت لبه ی بالا رونده و پایین رونده قابل پیکربندی است. نوشتن صفر در ISC2 باعث تریگر شدن این وقفه با لبه ی پایین رونده و نوشتن یک باعث تریگر شدن آن با لبه ی بالا رونده می شود.

هر یک از وقفه های خارجی دارای یک بیت پرچم هستند که در صورت تریگر شدن از پین وقفه ی خارجی و فعال بودن بیت مربوطه در رجیستر GICR و فعال بودن بیت فعال ساز وقفه (I)، علاوه بر یک شدن پرچم، می تواند باعث ایجاد وقفه شود. در این حالت پس از اجرای ISR پرچم آن وقفه به صورت سخت افزاری پاک می شود.

Bit	7	6	5	4	3	2	1	0
GIFR	INTF1	INTF0	INTF2					

روتین سرویس وقفه ها در CodeVision به صورت زیر تعریف می شود:

interrupt [شماره ی بردار وقفه] void نام روتین سرویس وقفه (void)

{

برنامه ی سرویس وقفه

}

شماره ی بردار وقفه ی در مورد ATMEGA16 عددی بین ۲ تا ۲۱ می باشد و می توان از نام معادل آن (جدول ابتدای فصل) نیز استفاده کرد.

مثال ۱:

```
#include <mega16.h>
#include <delay.h>

interrupt [2] void LED_ON(void)
{
        PORTA=0x01;
        delay_ms(1000);
        PORTA=0x00;
}

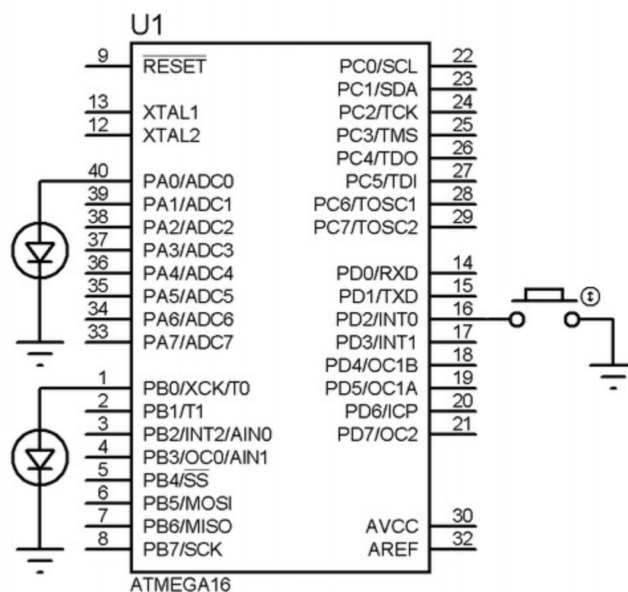
void main(void)
{
        DDRB=0xFF;
        PORTB=0x00;
        DDRA=0xFF;
        PORTA=0x00;
        DDRD=0x00;
```

```
    PORTD=0xFF;

    GICR=0b01000000;   // INT0: On
    MCUCR=0b00000010;  // INT0 Mode: Falling Edge

    #asm("sei")    // Global enable interrupts

    while (1)
    {
            PORTB=0x01;
            delay_ms(500);
            PORTB=0x00;
            delay_ms(500);
    };
}
```



- اگر وقفه ی خارجی به صورت حساس به لبه تنظیم شده باشد و در حال اجرای یکی از وقفه ها، وقفه ی دیگری اتفاق بیفتد، پس از خروج از ISR وقفه ی جاری به آن وقفه ها برحسب اولویت شان پاسخ داده خواهد شد.

- در صورتی که پین وقفه به صورت خروجی تعریف شود، آنچه روی پورت نوشته می شود می تواند باعث ایجاد وقفه شود، به این ترتیب می توان وقفه ی نرم افزاری ایجاد کرد.

- وقفه های حساس به سطح در پین INT0 و INT1 و همچنین وقفه ی حساس به لبه در INT2 به صورت آسنکرون تشخیص داده می شوند، بنابراین می توان از آن ها برای خارج کردن میکرو از همه ی Mode های کم مصرف استفاده نمود. حداقل عرض پالس در این حالت ۵۰ نانو ثانیه می باشد و برای مقادیر کمتر تشخیص وقفه ی خارجی تضمین نشده است.

- کامپایلر CodeVision به صورت پیش فرض SREG و رجیسترهایی CPU را که ممکن است در ISR تغییر پیدا کنند، قبل از اجرای روتین سرویس وقفه در Stack ذخیره (PUSH) کرده و پس از بازگشت از ISR آن ها را بازیابی (POP) می کند. برای غیر فعال کردن این قابلیت می توانید قبل از روتین سرویس وقفه از رهنود `#pragma savereg-` استفاده کنید.

پروژه ٦: آشکار ساز عبور از صفر

```
/****************************************************
Project : Zero Cross Detector
Author  : Reza Sepas Yar
Company : Pishro Noavaran Kavosh
****************************************************/

#include <mega16.h>
#include <delay.h>
#define xtal 4000000

interrupt [2] void switch_(void)
{
        PORTA=0x01;
        delay_ms(1);
        PORTA=0x00;
}

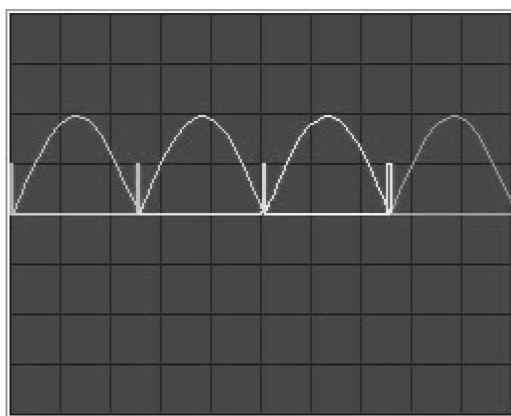
void main(void)
{
        DDRA=0xFF;
        PORTA=0x00;
        DDRD=0x00;
        PORTD=0xFF;

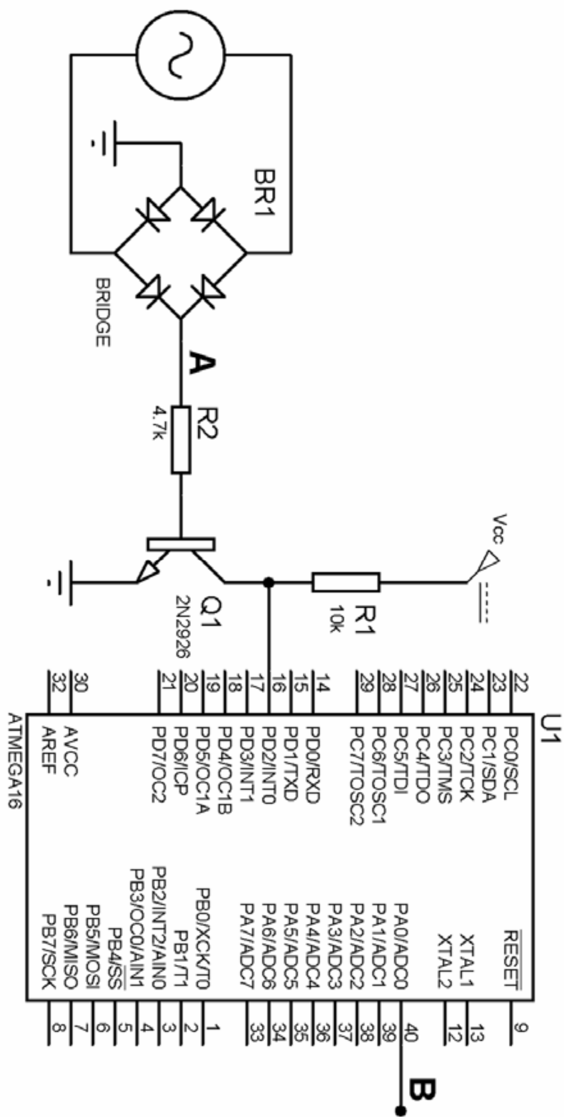
        GICR=0b01000000;    // INT0: On
        MCUCR=0b00000011;   // INT0 Mode: Rising Edge
```

```
        #asm("sei")    // Global enable interrupts

        while (1);
}
```

Output vs. Input:




BR1
BRIDGE
A
R2
4.7k
Q1
2N2926
R1
10k
Vcc
U1
ATMEGA16
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP
PD7/OC2
AVCC
AREF
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/XCK/T0
PB1/T1
PB2/INT2/AIN0
PB3/OC0/AIN1
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
B

تایمر/کانتر صفر

تایمر/ کانتر صفر یک تایمر ۸ بیتی می باشد کـه دارای چهـار Mode کـاری Normal، CTC، Fast PWM و Phase Correct PWM می باشد. پین T0 به عنوان ورودی کانتر و پـین OC0 خروجـی بخـش مقایسـه ی تایمر می باشد. این تایمر دارای سه رجیستر به نام های TCCR0، TCNT0 و OCR0 می باشد که به ترتیـب جهت پیکربندی تایمر، مقدار شمارنده و مقدار مقایسه استفاده مـی شـوند. همچـنین ایـن تـایمر در رجیسـترهای TIFR و TIMSK که به ترتیب رجیسترهای پرچم و وقفه تایمر می باشند با دیگر تایمرها مشترک می باشد.

مهم ترین رجیستر تایمر TTCR0 می باشد که بیت های Clock Select جهت انتخاب منبع کلاک تایمر و بیت های Wave Generation Mode برای تنظیم Mode کاری تایمر و بیت های Compare Match Output Mode پیکربندی پین OC0 را تعیین می کنند. عملکرد بیت FOC0 نیز بررسی خواهد شد.

TCCR0	7	6	5	4	3	2	1	0
نام بیت	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

Mode عملکرد تایمر با توجه به بیت های WGM01 و WGM00 به این ترتیب تعیین می شود:

Mode	WGM01	WGM00	عملکرد Mode
0	0	0	Normal
1	0	1	PWM, Phase Correct
2	1	0	Clear Timer on Compare Match (CTC)
3	1	1	Fast PWM

Noramal Mode ①

- **تایمر / کانتر ساده ی ۸ بیتی**

رجیسترهای TIMER0 :

TCCR0	7	6	5	4	3	2	1	0
نام بیت	FOC0	WGM00	COM01	COM00	WGM01	**CS02**	**CS01**	**CS00**
سطح منطقی	0	0	0	0	0	**X**	**X**	**X**

تایمر/کانتر صفر و یک دارای یک Prescaler مشترک بوده وضعیت منبع کلاک با توجه به بیت های Clock Select تعیین می شود:

CS02	CS01	CS00	وضعیت منبع کلاک تایمر
0	0	0	بدون کلاک (متوقف)
0	0	1	کلاک سیستم (بدون تقسیم)
0	1	0	۸/کلاک سیستم
0	1	1	٦٤/کلاک سیستم
1	0	0	٢٥٦/کلاک سیستم
1	0	1	١٠٢٤/کلاک سیستم
1	1	0	لبه ی پایین رونده ی پالس خارجی (T0)
1	1	1	لبه ی بالا رونده ی پالس خارجی (T0)

مقدار تایمر در هر لحظه از طریق رجیستر TCNT قابل خواندن و نوشتن است:

Bit	7	6	5	4	3	2	1	0
TCNT0	**TCNT0[7:0]**							

در زمان سرریز تایمر، بیت TOV0 از رجیستر TIFR یک می شود.

TIFR	7	6	5	4	3	2	1	0
نام بیت	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	**TOV0**
سطح منطقی	0	0	0	0	0	0	0	X

مثال۱: (تولید موج مربعی با $T = 512\ \mu s$):

```
#include<mega16.h>
#define xtal 8000000

void delay()
{
        TCCR0=0B00000010;        // Timer Clock = CLK/8
        while(!TIFR&0x01);       // Wait Until Overflow
        TIFR=TIFR|0B00000001;    // Clear TOV0
        TCCR0=0x00;              // Stop Timer0
}

void main()
{
        DDRA=0xFF;
        PORTA=0x00;
        TCCR0=0x00;
        TCNT0=0x00;

        while(1){
                PORTA.0=1;
                delay();
                PORTA.0=0;
                delay();
        }

}
```

مثال۲: (کاهش دوره ی تناوب به 400μs)

```
void delay()
{
        TCNT0=0x38;                    //TCNT=55
        TCCR0=0B00000010;
        while(!TIFR&0x01);
        TIFR=TIFR|0B00000001;
        TCCR0=0x00;
}
```

- ✓ در صورتی که بیت فعال ساز عمومی وقفه (I) فعال بوده و بیت های TOIE0 یا TOIE2 در رجیستر TIMSK یک باشند می توان با استفاده از وقفه، از سرریز شدن تایمر به عنوان وقفه استفاده کرد:

TIMSK	7	6	5	4	3	2	1	0
نام بیت	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
سطح منطقی	0	0	0	0	0	0	0	X

مثال۳: (موج مربعی $T = 400\mu s$ با استفاده از وقفه ی سرریز تایمر صفر)

```
#include <mega16.h>
#define xtal 8000000

interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
PORTA^=0xFF;
TCNT0=0x38;                    //TCNT=55
```

```
}

void main(void)
{
        DDRA=0xFF;
        PORTA=0x00;
        TCCR0=0B00000010;  // Timer Clock = CLK/8
        TIMSK=0x01;        //Enable TIMER0 Overflow
    Interrupt
        #asm("sei")        // Global enable interrupts
      TCNT0=0x38;

        while (1);
}
```

• تایمر/کانتر با عملکرد مقایسه

Bit	7	6	5	4	3	2	1	0
OCR0/2	**OCR0[7:0]**							

• محتوای رجیستر OCR0 به طور پیوسته با مقدار TCNT0 مقایسه می شود و در صورت برابـری باعـث تغییر وضعیت پین OC0 و یا وقفه ی تطابق می شود. در حالت برابری بیت OCF0 یا OCF1 یک شده و با فراخوانی سابروتین وقفه به صورت سخت افزاری صفر می شود. در صـورت عـدم اسـتفاده از وقفـه کاربر می تواند با نوشتن یک روی این بیت آن را پاک کند.

TIFR	7	6	5	4	3	2	1	0

نام بیت	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	**OCF0**	**TOV0**
سطح منطقی	0	0	0	0	0	0	X	X

تغییر وضعیت پین OC0 بوسیله بیت های COM00 و COM01 می باشد:

TCCR0	7	6	5	4	3	2	1	0
نام بیت	FOC0	WGM00	**COM01**	**COM00**	WGM01	**CS02**	**CS01**	**CS00**
سطح منطقی	X	0	X	X	0	**X**	**X**	**X**

COM01	COM00	وضعیت پین OC0
0	0	غیر فعال (I/O معمولی)
0	1	Toggle در وضعیت تطابق
1	0	Clear در وضعیت تطابق
1	1	Set در وضعیت تطابق

- در صورت یک کردن بیت FOC0 یا FOC1 به صورت آنـی مقـدار رجیسـتر TCNT0 بـا مقـدار OCR0 مقایسه شده و در صورت تطبیق مقایسه یک تغییر وضعیت روی پین OC0 ایجاد می شود. در این وضـعیت بیـت OCF0 یا OCF1 یک نشده و باعث ایجاد وقفه نیز نخواهد شد.
- در تمام حالت هایی که روی پین های OC شکل موج ایجاد می شود باید این پین به صورت خروجی تعریف شده باشد.

مثال٤: (ایجاد پالس مربعی با T = 512 μs روی پین OC0)

```
#include<mega16.h>
#define xtal 8000000

void main()
{
        DDRB=0xFF;
        PORTB=0x00;
        TCNT0=0x00;
        TCCR0=0B00010010; //toggle OC0 on compare match
        OCR0=0x63; //OCR0=99

        while(1);
}
```

برای استفاده از وقفه باید علاوه بر یک بودن فعال ساز عمومی وقفه ها، بیت فعال ساز مقایسه ی وقفه نیز Set شود.

TIMSK	7	6	5	4	3	2	1	0
نام بیت	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
سطح منطقی	X	X	0	0	0	0	X	X

در این حالت ISR به صورت زیر تعریف می شود:

```
interrupt [TIM0_COMP] void timer0_comp_isr(void)
{
زیر برنامه ی سرویس وقفه
}
```

CTC Mode ②

در این Mode تایمر همانند وضعیت نرمال با عملکرد مقایسه عمل می کند با این تفاوت که در زمان تطابق رجیسترهای OCR0 و TCNT0 مقدار رجیستر TCNT0 صفر شده و در واقع بر خلاف حالت قبل مقدار ماکزیمم TCNT0 عدد موجود در رجیستر OCR0 می باشد. مقدار بیت های WGM00 و WGM01 در این Mode به ترتیب برابر ۰ و ۱ می باشد.

TCCR0	7	6	5	4	3	2	1	0
نام بیت	FOC0	WGM00	**COM01**	**COM00**	WGM01	**CS02**	**CS01**	**CS00**
سطح منطقی	X	0	X	X	1	**X**	**X**	**X**

- در این حالت فرکانس موج ایجاد شده روی پین OC0 از رابطه ی زیر بدست می آید:

$$f_{OC0} = \frac{f_{CLK_I/O}}{2.N.(1+OCR0)}$$

مثال۵: (ایجاد موج مربعی با فرکانس 5KHz روی پین OC0):

```
#include<mega16.h>
#define xtal 8000000

void main()
{
        DDRB=0xFF;
        PORTB=0x00;
        TCNT0=0x00;
        OCR0=0x63; //OCR0=99
        TCCR0=0B00011010; //toggle OC0 on compare match

        while(1);
}
```

$$f_{OC0} = \frac{8000000}{2.8.(1+99)} = 5000 Hz$$

وضعیت بیت های فعال ساز وقفه ی سرریز و وقفه ی مقایسه در CTC Mode همانند وضعیت نرمال می باشد. با یک بودن بیت OCIE0 وقفه ی مقایسه فعال می باشد و می توان در ISR این وقفه مقدار OCR0 را تغییر داد.

- مقدار دهی به رجیستر OCR0 باید با دقت انجام شود زیرا در Mode های غیر PWM این رجیستر دارای بافر دوبل نمی باشد. وجود بافز دوبل باعث می شود که اگر OCR0 به مقداری کمتر از TCNT0 تغییر کند،

برای از دست نرفتن مقایسه ی فعلی مقدار جدید در بافر دوبل ذخیره شده و پس از سرریز این مقدار جدیـد در OCR0 بارگذاری شود.

پروژه ۷: فرکانس متر دیجیتال

```
/*****************************************************
Project : Frequency Meter
Author  : Reza Sepas Yar
Company : Pishro Noavaran Kavosh
*****************************************************/

#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#include <lcd.h>
#define xtal 8000000

#asm
   .equ __lcd_port=0x1B ;PORTA
#endasm

unsigned long int timer0_ov;
unsigned long int in_freq;
unsigned char lcd_buff[20];

interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
```

```
timer0_ov ++;
}

void main(void)
{

// Timer/Counter 0 initialization
// Clock source: T0 pin Falling Edge
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCNT0=0x00;
OCR0=0x00;
TCCR0=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;

// LCD module initialization
lcd_init(16);

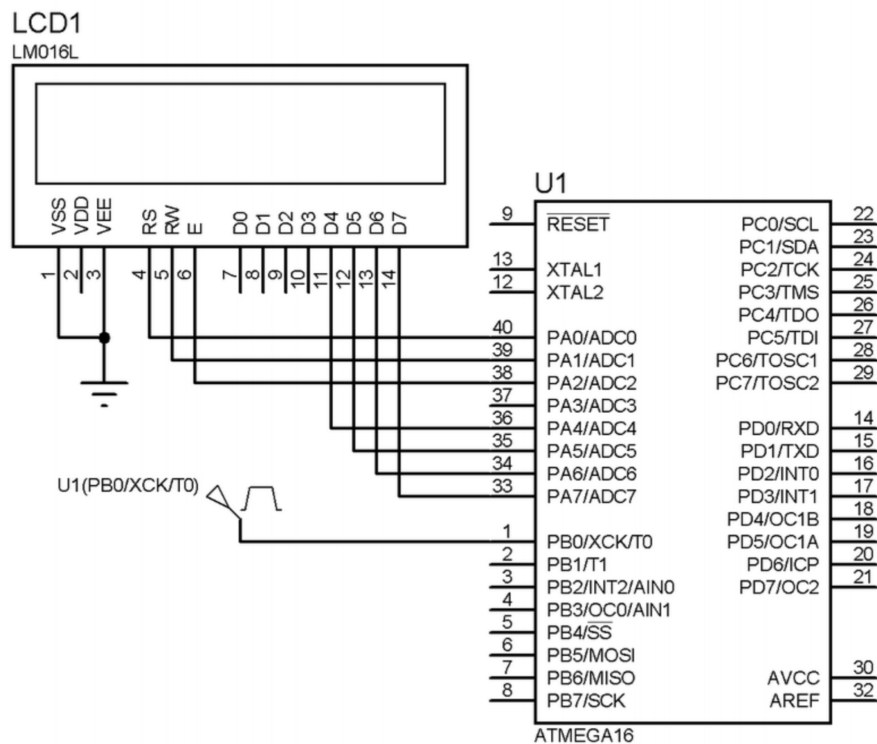
while (1)
{
      TCCR0=0x06;     // Start Timer T0 pin Falling Edge

 #asm("sei")      // Global enable interrupts

      delay_ms(1000);

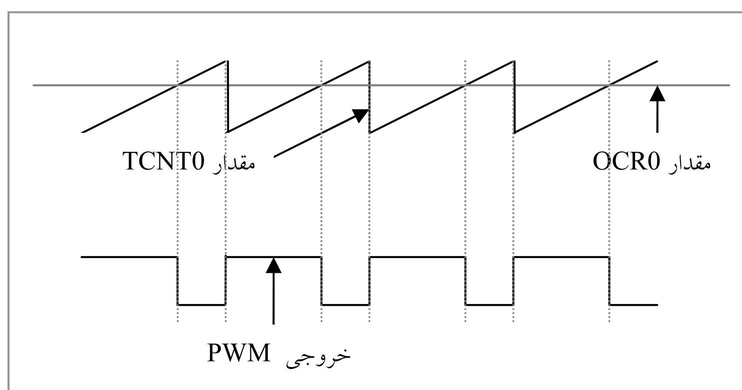
      #asm("cli");    // Global disable interrupts
```

```
    in_freq = timer0_ov * 256 + TCNT0;
    sprintf(lcd_buff,"Freqency=%d",in_freq);
    lcd_clear();
    lcd_puts(lcd_buff);
    TCCR0=0x00;     //Stopt Timer0
    timer0_ov=0;    //Prepare for next count
    TCNT0=0;        //Clear Timer0
    };
}
```



Fast PWM Mode ③

این حالت مشابه Mode نرمال می باشد با ایـن تفـاوت کـه پـین OC0 فقـط در حالـت برابـری رجیسـترهای TCNT0و OCR0 تغییر حالت نمی دهد، بلکه در زمان سرریز رجیستر TCNT0 نیز مقدار این پین تغییر مـی کند.



مقدار بیت های WGM00 و WGM01 در این Mode برابر ۱ می باشد.

Bit	7	6	5	4	3	2	1	0
TCCR0	0	1	COM01	COM00	1	CS02	CS01	CS00

در Mode های PWM عملکرد بیت های COM00 و COM01 متفاوت از دو وضعیت قبلـی و بـه صـورت زیر می باشد:

COM01	COM00	وضعیت پین OC0

0	0	غیر فعال (I/O معمولی)
0	1	رزرو شده
1	0	Clear دروضعیت تطابق، Set در زمان سرریز (PWM غیر معکوس)
1	1	Set در وضعیت تطابق، Clear در زمان سرریز (PWM معکوس)

برای محاسبه ی فرکانس موج PWM تولید شده می توان از فرمول زیر استفاده نمود:

$$f_{PWM} = \frac{f_{clk_I/O}}{N.X}$$

N = Prescale = 1, 8, 64, 256, 1024 X = ۲۵٦ – مقدار بارگذاری شده در تایمر

✓ از وقفه ی سرریز تایمر می توان برای مقدار اولیه دادن به TCNT0 و یا تغییر مقدار OCR0 استفاده نمود، اگرچه بهتر است مقدار OCR0 در روتین وقفه ی مقایسه تغییر داده شود.

✓ با مقدار اولیه دادن به TCNT0 می توان فرکانس موج PWM را تغییر داد.)

مثال٦: (تولید موج PWM با فرکانس 4KHz و زمان وظیفه ی ۲۰ درصد)

```
#include <mega16.h>
#define xtal 8000000


interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
TCNT0=0x06;
}

void main(void)
{
```

```
PORTB=0x00;
DDRB=0x08;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 1000.000 kHz
// Mode: Fast PWM top=FFh
// OC0 output: Non-Inverted PWM
TCCR0=0x6A; //0x7A for inverted PWM
TCNT0=0x06;
OCR0=0x38;  //OCR0 = 56

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;

// Global enable interrupts
#asm("sei")

while (1);

}
```

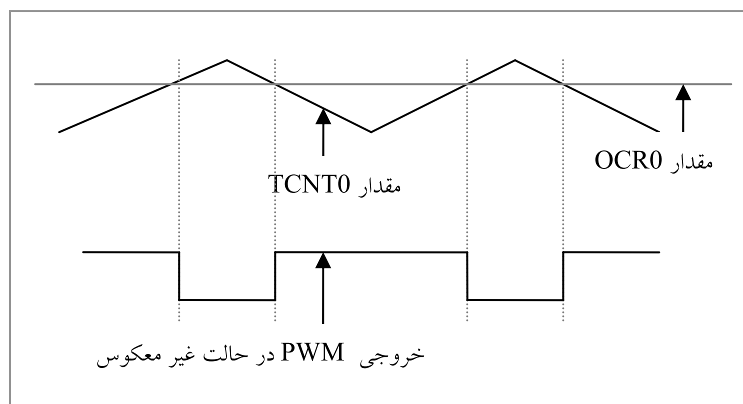
$$f_{PWM} = \frac{٨٠٠٠٠٠٠}{٨(٢٥٦-٦)} = ٤٠٠٠\text{Hz}$$

$$\text{DutyCycle} = \frac{\text{OCR}٠}{٢٥٠} \times ١٠٠\% = \frac{٥٠}{٢٥٠} \times ١٠٠\% = ٢٠\%$$

- با کمتر شدن OCR0 زمان وظیفه کمتر شده و تا حدی که مقدار صفر یک پالس سوزنی به عرض یک سیکل ایجاد خواهد کرد.
- با مقداردهی ۲۵۶ به OCR0 مقدار مقایسه و سرریز برابر شده و پالس خروجی بسته به مقدار COM00 و COM01 همواره صفر یا یک خواهد بود.

Phase Correct PWM Mode ④

این Mode شبیه حالت Fast PWM می باشد با این تفاوت که تایمر به صورت دو شیبه (Dual Slope) عمل شمارش را انجام می دهد. به اینصورت که تایمر از عدد صفر شروع به شمارش کرده و به صورت افزایشی تا عدد 0xFF افزایش می یابد و بعد از رسیدن به این مقدار عدد موجود در بافر OCR0 در رجیستر OCR0 بارگذاری می شود. بعد از این لحظه جهت شمارش تایمر عوض شده و به صورت کاهشی تا عدد صفر می شمارد با رسیدن به این عدد پرچم سرریز تایمر یک شده و در صورت یک بودن بیت فعال ساز وقفه، برنامه به ISR سرریز تایمر منشعب شده و بیت پرچم وقفه بوسیله سخت افزار صفر می شود. مسئله ی مهم این است که در هر دو حالت شمارش افزایشی و کاهشی عمل مقایسه بین رجیسترهای TCNT0 و OCR0 انجام می گیرد و در صورت برابری پرچم OCF0 یک شده و باعث تغییر در پین OC0 می شود.



تغییر پین OC0 مطابق جدول زیر می باشد:

COM01	COM00	وضعیت پین OC0
0	0	غیر فعال (I/O معمولی)
0	1	رزرو شده
1	0	Clear در وضعیت تطابق، در زمان شمارش افزایشی (PWM غیر معکوس) Set در وضعیت تطابق، در زمان شمارش کاهشی
1	1	Set در وضعیت تطابق، در زمان شمارش افزایشی (PWM معکوس) Clear در وضعیت تطابق، در زمان شمارش کاهشی

- در صورت تغییر مقدار رجیستر OCR0 مقدار جدید در بافر این رجیستر نوشته می شود و در زمان رسیدن به 0xFF رجیستر OCR0 به روز می شود.
- پرچم سرریز تایمر صفر زمانی فعال می شود که رجیستر TCNT0 برابر صفر شود و نه 0xFF، بنابراین باید دقت داشت که اگر در زمان شروع به کار مقدار این رجیستر صفر باشد پرچم سرریز فعال خواهد شد.
- فرکانس PWM در حالت Phase Correct از رابطه ی زیر قابل محاسبه است:

$$f_{PWM} = \frac{f_{clk_I/O}}{N \times ٥١٠} \qquad N = 1, 8, 64, 256, 1024$$

- رابطه ی بالا نشان می دهد فرکانس موج PWM ثابت است و ارتباطی به رجیسترهای OCR0 و TCNT0 ندارد.

- در حالت PWM غیر معکوس با افزایش مقدار OCR0 مقدار متوسط موج PWM افزایش یافته و با کاهش آن مقدار متوسط کاهش می یابد و در حالت PWM معکوس، عکس این قضیه صحیح است.

پروژه ۸: کنترل موتور DC با PWM

```
/*****************************************************
Project : DC Motor Control
Author  : Reza Sepas Yar
Company : Pishro Noavaran Kavosh
*****************************************************/

#include <mega16.h>
#define xtal 1000000

char digits[8]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07};
unsigned char;
unsigned char p_state;
unsigned char key;
unsigned char i;

void main(void)
{

PORTB=0x00;
DDRB=0xFF;
DDRD = 0xFF;
PORTD = digits[0];
DDRC = 0x00;
```

```
PORTC = 0xFF;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 15.625 kHz
// Mode: Phase correct PWM top=FFh
// OC0 output: Non-Inverted PWM
TCCR0=0x63;
TCNT0=0x00;
OCR0=10;

while (1)
{
        if(!PINC.0)
    {
            if(key!=p_state)
            {
                    if(i==7)
                    {
                            i=0;
                        PORTC=digits[0];
                 }
                 else
                        i++;

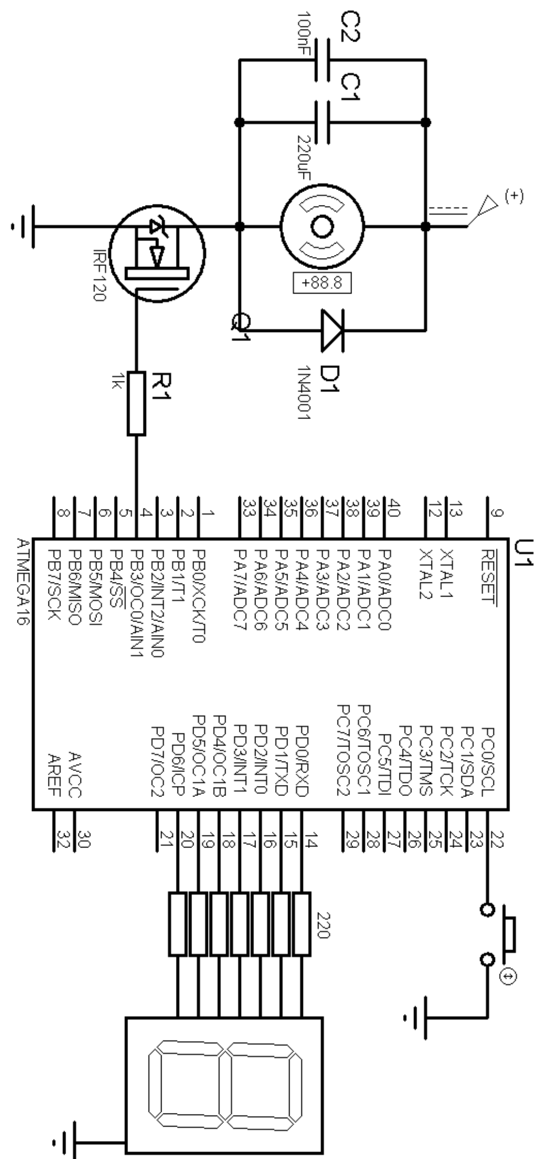
                    PORTD = digits[i];
                    OCR0 = i*10+10;
                 p_state=key;
            };
    }
```

```
    else

              p_state=0xFF;


        };

}
```



توضیح: برای آزمایش برنامه به جای موتور می توان از LED استفاده نمود.

عملکرد تایمر دو

به طور کلی عملکرد تایمر ۲ مشابه تایمر صفر می باشد و رجیسترهای مربوطه با همان نـام و دارای پسـوند ۲ مـی باشند، با این تفاوت که تایمر۲ برخلاف تایمرهای صفر و یک نمی تواند از پایه خارجی T0 یا T1 کلاک دریافت کند و در عوض می توان با وصل کردن یک کریستال ۳۲.۷٦۸ کیلوهرتز به پین هـای TOSC1 و TOSC2 از آن در وضعیت آسنکرون جهت RTC استفاده نمود. از آنجایی که تایمر۲ دارای Prescaler مجـزا از دو تـایمر۰ و۱ می باشد با تقسیم کریستال ۳۲۷٦۸ هرتز بر ۱۲۸ می توان به زمان سرریز ۱ ثانیه که مناسب بـرای عملکـرد سـاعت است دست پیدا کرد. تنظیمات Prescale برای این تایمر به صورت زیر می باشد:

وضعیت منبع کلاک تایمر	CS00	CS01	CS02
بدون کلاک (متوقف)	0	0	0
کلاک سیستم (بدون تقسیم)	1	0	0
۸/کلاک سیستم	0	1	0
۳۲/کلاک سیستم	1	1	0
٦٤/کلاک سیستم	0	0	1
۱۲۸/کلاک سیستم	1	0	1
۲٥٦/کلاک سیستم	0	1	1
۱۰۲٤/کلاک سیستم	1	1	1

پیکر بندی RTC با رجیستر وضعیت آسنکرون یا ASSR انجام می شود:

Bit	7	6	5	4	3	2	1	0
ASSR					AS0	TCN2UB	OCR2UB	TCR2UB

Asynchronous Timer/Counter2: با Set کردن این بیت منبع کلاک تایمر ۲ از کلاک سیستم به کریستال خارجی در پایه های TOSC1 و TOSC2 تغییر می کند.با تغییر دادن این بیت ممکن است مقدار رجیسترهای TCNT2، OCR2 و TCCR2 خراب شود.

Timer/Counter2 Update Busy: برای تضمین عملکرد صحیح در وضعیت آسنکرون رجیسترهای تایمر۲ برخلاف تایمر ۰ و ۱ به صورت بافر شده بروز میشوند. بدین ترتیب که وقتی روی رجیستر TCNT2 مقداری نوشته شود، بیت TCN2UB یک می شود و مقداری که در رجیستر موقتی ذخیره شده است به TCNT2 منتقل می شود. با اتمام بروز رسانی TCNT2 این بیت توسط سخت افزار صفر می شود. صفر بودن OCR2UB نشان دهنده ی آمادگی TCNT2 برای پذیرفتن مقدار جدید است.

Output Compare Register2 Update Busy: این بیت همانند TCN2UB بوده با این تفاوت که بر روی رجیستر OCR2 عمل می کند.

Timer/Counter Control Register2 Update Busy: این بیت همانند TCN2UB بوده با این تفاوت که بر روی رجیستر TCCR2 عمل می کند.

- در حالتی که پرچم مشغول بودن یک رجیستر یک می باشد، نوشتن بر روی آن رجیستر باعث می شود که مقدار بروز شده صحیح نباشد و ممکن است باعث وقفه ی ناخواسته شود.

- مکانیسم خواندن این سه رجیستر متفاوت می باشد، بدین صورت که زمان خواندن TCNT2 مقدار خـود رجیستر خوانده شده و با خواندن OCR2 و TCCR2 مقدار موجود در رجیستر موقت خوانده می شود.
- تایمر۲ در وضعیت آسنکرون در حالت Power-Save نیز فعـال بـوده و پـس از سـرریز شـدن تـایمر از وضعیت Power-Save خارج شده و در وصرت فعال بودن وقفـه، ISR را اجـرا نمـوده و مجـددا وارد حالت Power-Save می شود.

پروژه ۹: ساعت با RTC میکروکنترلر

```
/*****************************************************
Project : Real Time Clock
Author  : Reza Sepas Yar
Company : Pishro Noavaran Kavosh
*****************************************************/

#include <mega16.h>
#include <lcd.h>
#include <stdio.h>
#define xtal 8000000

#asm
   .equ __lcd_port=0x1B ;PORTA
#endasm

unsigned char second, minute,hour;
unsigned char lcd_buff[10];
```

```
interrupt [TIM2_OVF] void timer2_ovf_isr(void)
{
        if(second==59)
        {
                second=0;
                if(minute==59)
                {
                        minute=0;
                        if(hour==23)
                                hour=0;
                        else
                                hour++;
                }
                else
                        minute++;
        }
        else
                second++;

        sprintf(lcd_buff,"Time = %d:%d:%d",hour, minute,
    second);
        lcd_clear();
        lcd_puts(lcd_buff);
}

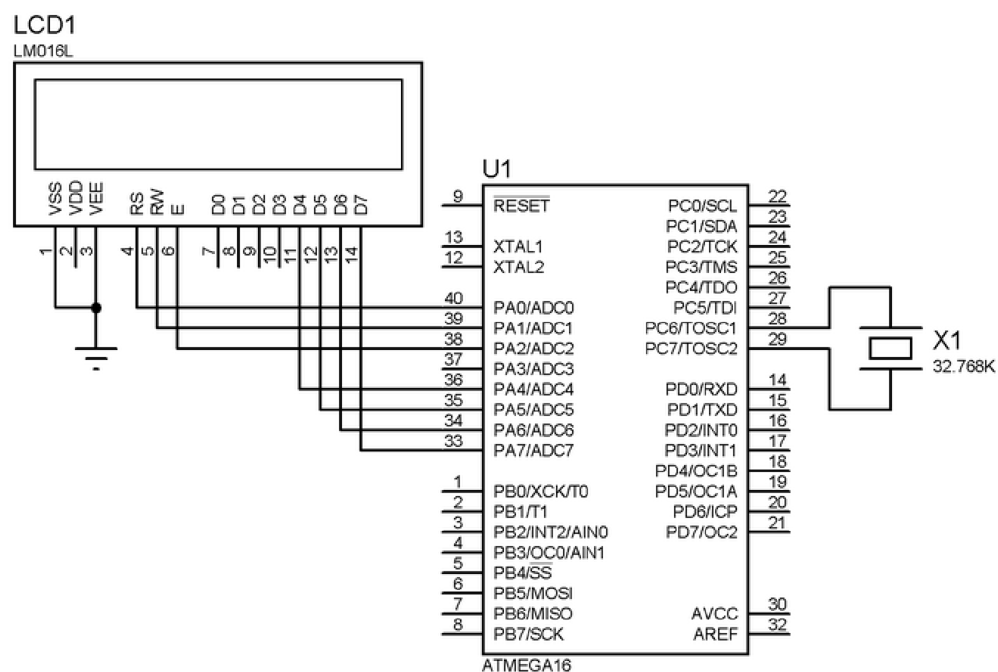
void main(void)
{
// Clock source: TOSC1 pin
// Clock value: PCK2/128
```

```
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x08;
TCCR2=0x05;
TCNT2=0x00;
OCR2=0x00;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x40;

lcd_init(16);

#asm("sei") // Global enable interrupts

while (1);
}
```



تایمر/کانتر یک

تایمر یک تایمری ۱٦ بیتی است و در آن علاوه بر امکانات تایمر صفر، یک بخش دیگر به نام بخش Capture به آن افزوده شده است. این بخش در زمان های خاص، عدد شمارش شده توسط تایمر یک و زمان سپری شده را ثبت کرده و از طریق آن امکان اندازه گیری های زمانی را فراهم می آورد. تایمر یک دارای پنج Mode کاری به نام های Normal، CTC، Fast PWM Mode، Phase Correct PWM Mode و Phase and Frequency Correct می باشد. Mode های PWM در تایمر ۱ بسیار متنوع و دارای ۱۲ حالت PWM می باشد. در این تایمر پین T1 به عنوان ورودی کانتر و پین های OC1A و OC1B به عنوان خروجی مقایسه گر عمل می کنند. همچنین پین ICP1 برای ورودی بخش Capture تایمر یک می باشد.

به علت ۱٦ بیتی بودن تایمر، رجیسترهای TCNT1 و OCR1A و OCR1B شانزده بیتی می باشند که هر کدام دارای دو بایت L و H هستند. همچنین تایمر یک دارای دو واحد مقایسه ی مجزا می باشد که مقدار موجود در رجیسترهای OCR1A و OCR1B را با TCNT1 مقایسه کرده و در صورت برابری وضعیت پین های OC1A و OC1B را تغییر می دهند. همچنین رجیستر ICR1 نیز که رجیستر واحد Capture است رجیستری ۱٦ بیتی می باشد.

رجیسترهای شانزده بیتی تایمر ۱: TCNT1، OCR1A، OCR1B و ICR1

رجیسترهای ۸ بیتی TCCR1A و TCCR1B کنترل تایمر را بر عهده دارند:

TCCR1A	7	6	5	4	3	2	1	0
نام بیت	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

TCCR1B	7	6	5	4	3	2	1	0
نام بیت	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

Mode کاری تایمر بوسیله ی بیت های WGM10, WGM11, WGM12 و WGM13 تعیین می شود:

	WGM13	WGM12	WGM11	WGM10	Mode کاری	TOP	TOV=1
۰	۰	۰	۰	۰	Normal	0xFFFF	0xFFFF
۱	۰	۰	۰	۱	PWM, Phase Correct, 8-bit	0x00FF	0
۲	۰	۰	۱	۰	PWM, Phase Correct, 9-bit	0x01FF	0
۳	۰	۰	۱	۱	PWM, Phase Correct, 10-bit	0x03FF	0
٤	۰	۱	۰	۰	CTC	OCR1A	0xFFFF
٥	۰	۱	۰	۱	Fast PWM, 8-bit	0x00FF	TOP
٦	۰	۱	۱	۰	Fast PWM, 9-bit	0x01FF	TOP
۷	۰	۱	۱	۱	Fast PWM, 10-bit	0x03FF	TOP
۸	۱	۰	۰	۰	PWM, Phase and Frequency Correct	ICR1	0
۹	۱	۰	۰	۱	PWM, Phase and Frequency Correct	OCR1A	0
۱۰	۱	۰	۱	۰	PWM, Phase Correct	ICR1	0
۱۱	۱	۰	۱	۱	PWM, Phase Correct	OCR1A	0
۱۲	۱	۱	۰	۰	CTC	ICR1	0xFFFF
۱۳	۱	۱	۰	۱	رزرو شده	-	-
۱٤	۱	۱	۱	۰	Fast PWM	ICR1	TOP

۱۵	۱	۱	۱	۱	Fast PWM	OCR1A	TOP

- تعریف TOP: تایمر وقتی به مقدار TOP می رسد که برابر با بالاترین مقدار در رشته ی شمارش خود است. این مقدار می تواند مقادیر ثابتی مثل 0x03FF، 0x01FF و یا 0x00FF بوده و یا مقدار نگهداری شده در یکی از رجیسترهای OCR1A یا ICR1 باشد.

FOC1A و FOC1B: بیت های Force بخش مقایسه گر که عملکرد آن ها همانند FOC0 در تایمر صفر و دو می باشد. به این ترتیب که در Mode های غیر PWM، یک کردن این بیت بدون اینکه وقفه ای ایجاد کند در صورت تطبیق مقایسه، باعث تغییر وضعیت پین های OC1A و OC1B مطابق با وضعیت بیت های COM در TCCR1 می شود.

بیت های COM1A0، COM1A1، COM1B0 و COM1B1: تغییر وضعیت پین های OC1A و OC1B را در حالت تطبیق معین می کنند که مقدار آن ها بسته به Mode کاری عملکرد متفاوتی را ایجاد می کند. بنابراین در بررسی Mode های مختلف آن را مطالعه خواهیم کرد.

بیت های CS10، CS11 و CS12: برای تعیین منبع کلاک تایمر می باشند:

CS02	CS01	CS00	وضعیت منبع کلاک تایمر
0	0	0	بدون کلاک (متوقف)
0	0	1	کلاک سیستم (بدون تقسیم)
0	1	0	۸/کلاک سیستم

0	1	1	٦٤/کلاک سیستم
1	0	0	٢٥٦/کلاک سیستم
1	0	1	١٠٢٤/کلاک سیستم
1	1	0	لبه ی پایین رونده ی پالس خارجی (T1)
1	1	1	لبه ی بالا رونده ی پالس خارجی (T1)

بیت ICES1: بیت تعیین لبه ی ورودی بخش Capture از پین ICP1. با صفر بودن این بیت لبه ی پایین رونده و با یک بودن آن لبه ی بالا رونده باعث تریگر می شود.

بیت ICNC1: بیت فعال ساز حذف نویز در ورودی پین ICP1

نتایج حاصل از کارکرد تایمر۱ در ٤ بیت از رجیستر TIFR به نام های TOV1 (پرچم سرریز) OCF1A (پرچم تطابق مقایسه گر A) OCF1B (پرچم تطابق مقایسه گر B) و ICF1 (پرچم بخش Capture تایمر۱) منعکس می شوند:

TIFR	7	6	5	4	3	2	1	0
نام بیت	OCF2	TOV2	**ICF1**	**OCF1A**	**OCF1B**	**TOV1**	OCF0	TOV0
سطح منطقی	0	0	X	X	X	X	0	0

یک شدن هر یک از این پرچم ها در صورت فعال بودن بیت فعال ساز عمومی (I) و فعال بودن وقفه ی مربوطه در رجیستر TIMSK می تواند باعث انشعاب برنامه به ISR مربوط به آن وقفه شود:

TIMSK	7	6	5	4	3	2	1	0
نام بیت	OCIE2	TOIE2	**TICIE1**	**OCIE1A**	**OCIE1B**	**TOIE1**	OCIE0	TOIE0
سطح منطقی	0	0	X	X	X	X	0	0

با اجرا شدن ISR به صورت خودکار بیت پرچم وقفه صفر شده و یا می تواند با نوشتن یک روی آن بوسیله ی نرم افزار آن را پاک کرد.

Normal Mode ①

این Mode همانند مشابه آن در تایمر صفر می باشد با این تفاوت که تایمر تا عدد 0xFFFF شمارش کرده و با رسیدن به آن تایمر سرریز کرده و بیت TOV1 یک شده و در صورت فعال بودن وقفه می تواند باعث اجرای ISR مربوطه شود. در Mode عادی هر دو مقایسه گر A و B فعال بوده و هر کدام به طور مستقل عمل مقایسه را روی رجیستر TCNT1 و OCR1A و OCR1B انجام می دهند. در صورت برابری بیت OCF1A یا OCF1B یک شده و خروجی OC1A یا OC1B مطابق جدول زیر تغییر وضعیت داده و در صورت فعال بودن وقفه می توانند باعث ایجاد وقفه شوند.

COM1A1/COM1B1	COM1A0/COM1B0	وضعیت پین OC1A یا OC1B
0	0	غیر فعال (I/O معمولی)
0	1	Toggle در وضعیت تطابق
1	0	Clear در وضعیت تطابق
1	1	Set در وضعیت تطابق

- در صورت استفاده از OC1A یا OC1B برای تولید شکل موج، باید این پین ها به صورت خروجی پیکربندی شوند.

مثال ۷: (تولید دو شکل موج با دوره تناوب ۱۳۱ میلی ثانیه و اختلاف فاز ۱۰ میلی ثانیه)

```
#include <mega16.h>
#define xtal 8000000

void main(void)
{

PORTD=0x00;
DDRD=0x30;

// Mode: Normal top=FFFFh
TCCR1A=0x50;  //toggle OC1A & OC1B
TCCR1B=0x02;  //Clock/8
OCR1AH=0x00;
OCR1AL=0xFF;  //OCR1A=255
```

```
OCR1BH=0x28;
OCR1BL=0x0F;  //OCR1B=10255

while (1);

}
```

$T = ۲ \times ۲^{۱۶} \times ۱\mu s = ۱۳۱۰۷۲\mu s = ۱۳۱ms$

اختلاف فاز $= OCR۱B - OCR۱A = ۱۰۲۵۵ - ۲۵۵ = ۱۰۰۰۰\mu s = ۱۰ms$

CTC Mode ②

در این حالت مقدار رجیستر TCNT1 به طور پیوسته با مقدار رجیستر OCR1A یا ICR1 مقایسه می شود و در صورت برابری مقدار رجیستر TCNT1 برابر صفر می شود. بنابراین در این حالت مقدار TOP تایمر را با توجه به مقدار موجود در بیت های WGM مقدار رجیسترهای OCR1A یا ICR1 تعیین می کنند.

با رسیدن تایمر به مقدار TOP خود بر حسب اینکه مقدار ماکزیمم OCR1A یا ICR1 انتخاب شده باشد به ترتیب پرچم های OCF1A یا ICF1 یک شده و در صورت فعال بودن وقفه از آن می توان برای تغییر دادن مقدار مقایسه استفاده کرد. این عمل باید با دقت صورت گیرد زیرا رجیستر مقایسه ی تایمرها فقط در Mode های PWM دارای بافر دوبل می باشند. در این حالت فرکانس موج ایجاد شده روی پایه های OC1A یا OC1B مطابق رابطه ی زیر می باشد:

$$f_{OC۱x} = \frac{f_{CLK_I/O}}{۲.N.(۱ + OCR۱A)}$$

مثال۸: (تولید موج مربعی با فرکانس ۱ کیلوهرتز روی پایه ی OC1A)

```
#include <mega16.h>
#define xtal 8000000

void main(void)
{

PORTD=0x00;
DDRD=0x30;

// Mode: CTC top=01F3h
TCCR1A=0x40;
TCCR1B=0x0A;
OCR1AH=0x01;
OCR1AL=0xF3;  //OCR1A=499

while (1);
}
```

$$f = \frac{۸۰۰۰۰۰۰}{۲ \times ۸(۱+۴۹۹)} = ۱۰۰۰ = ۱KHz$$

Fast PWM Mode ②

بر خلاف تایمرهای صفر و دو که در آن موج های PWM تولید شده دارای دقت ثابت ۸ بیتی هستند، تایمر ۱ قادر است سیگنال های PWM ای با دقت متغیر را ارائه کند، این مسئله باعث می شود که کاربر بتواند علاوه بر تغییر Duty Cycle فرکانس موج را به صورت سخت افزاری کنترل کند (بدون مقدار اولیه دادن به TCNT1)

PWM سریع دارای پنج Mode می باشد: (WGM1[3:0] = ۱۵ ،۱٤ ،۷ ،٦ ،۵)

۱. PWM سریع ۸ بیتی (0xFF = TOP)
۲. PWM سریع ۹ بیتی (0x1FF = TOP)
۳. PWM سریع ۱۰ بیتی (0x03FF = TOP)
٤. PWM سریع با ICR1 = TOP
۵. PWM سریع با OCR1A = TOP

در این Mode تایمر از صفر تا مقدار TOP خود شروع به شمارش کرده و پس از از سرریز مجددا از صفر شروع به کار می کند. در صورتی که مقایسه ی خروجی در حالت PWM غیر معکوس باشد در حالت تطبیق مقایسه بین رجیسترهای TCNT1 و OCR1x پین OC1x یک شده و با رسیدن به مقدار TOP پاک می شود. در صورتی که خروجی PWM معکوس باشد وضعیتی عکس وجود خواهد داشت. دقت موج PWM خروجی می تواند مقادیر ثابت ۸، ۹ یا ۱۰ بیتی داشته ویا بوسیله ی رجیسترهای ICR1 یا OCR1A به مقدار دلخواه تنظیم

شود. در این حالت حداقل مقدار مجاز ۲ بیت (با دادن مقدار 0x0003 به رجیسترهای ICR1 یا OCR1x) و حداکثر آن ۱٦ بیت می باشد (با دادن مقدار 0xFFFF به رجیسترهای ICR1 یا OCR1x).

دقت موج PWM بر حسب مقدار ماکزیمم از رابطه ی زیر به دست می آید:

$$resolution = \frac{\log(TOP + ۱)}{\log(۲)}$$

با رسیدن تایمر به مقدار TOP پرچم سرریز TOV1 فعال شده و با تطبیق مقایسه نیز بیت OCF1A یا OCF1B یک می شود. در این حالت ها اگر وقفه ی مربوطه فعال شده باشد می توان در ISR آن وقفه مقدار مقایسه را تغییر داد. باید توجه داشت که مقدار رجیسترهای مقایسه باید از مقدار TOP کمتر باشد در غیر اینصورت هیچگاه مقایسه ای صورت نمی گیرد.

تغییر وضعیت پین های OC1A و OC1B در حالت تطبیق مقایسه و سرریز مطابق جدول زیر خواهد بود:

COM1A1 /COM1B1	COM1A0 /COM1B0	وضعیت پین OC1A یا OC1B
0	0	غیر فعال (I/O معمولی)
0	1	اگر WGM1[3:0] = 15 باشد: Toggle پین OC1A در وضعیت تطابق و OC1B پین I/O معمولی برای دیگر حالت های WGM1[3:0]: غیر فعال (I/O معمولی)
1	0	Clear در وضعیت تطابق و Set در وضعیت TOP (PWM غیر معکوس)
1	1	Set در وضعیت تطابق و Clear در وضعیت TOP (PWM معکوس)

فرکانس موج PWM حاصل از رابطه ی زیر بدست می آید:

$$f_{PWM} = \frac{f_{Clk_I/O}}{N.(1 + TOP)}$$

مثال۹: (موج PWM با فرکانس ۱ کیلو هرتز و زمان وظیفه ی ۲۵ درصد)

```
#include <mega16.h>
#define xtal 8000000

void main(void)
{

PORTD=0x00;
DDRD=0x20;

// Mode: Fast PWM top=03FFh
// OC1A output: Non-Inv.
// OC1B output: Disconnected
TCCR1A=0x83;
TCCR1B=0x0A; //10 Bit PWM
OCR1AH=0x00;
OCR1AL=0xFF;

while (1);
}
```

$$f_{PWM} = \frac{8000000}{8.(1 + 1023)} = 976 \approx 1KHz$$

$$DutyCycle = \frac{256}{1024} \times 100\% = 25\%$$

- با مقدار اولیه دادن به TCNT1 در ISR سرریز تایمر، می توان به فرکانس دقیق ۱ کیلوهرتز رسید:

```
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
TCNT1=24;
}

TIMSK=0x04; //Enable TOV1
#asm("sei") //Enable Interrupts
```

③ Phase Correct Mode

PWM تصحیح فاز دارای پنج Mode کاری می باشد: (۱، ۲، ۳، ۱۰، ۱۱ = WGM1[3:0])

۱. PWM تصحیح فاز ۸ بیتی (0xFF = TOP)
۲. PWM تصحیح فاز ۹ بیتی (0x1FF = TOP)
۳. PWM تصحیح فاز ۱۰ بیتی (0x03FF = TOP)
٤. PWM تصحیح فاز با ICR1 = TOP
٥. PWM تصحیح فاز با OCR1A = TOP

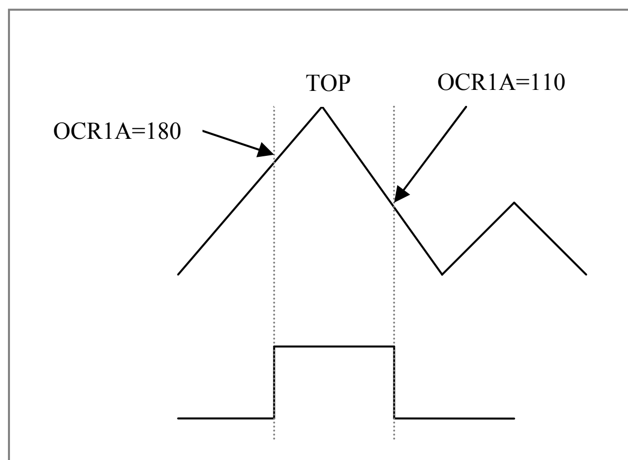
در این Mode تایمر به طور پیوسته از مقدار صفر تا TOP و از TOP تا صفر می شمارد. در حالت PWM غیر معکوس در حالی که تایمر به صورت صعودی می شمارد در لحظه ی برابری رجیسترهای TCNT1 و OCR1x پین OC1x صفر شده و در حالت شمارش نزولی با تطابق دو رجیستر این پین یک می شود. در حالت PWM معکوس، عکس این قضیه برقرار است.

دقت موج PWM خروجی می تواند مقادیر ثابت ۸، ۹ یا ۱۰ بیتی داشته ویا بوسیله ی رجیسترهای ICR1 یا OCR1A به مقدار دلخواه تنظیم شود. در این حالت حداقل مقدار مجاز ۲ بیت (با دادن مقدار 0x0003 به رجیسترهای ICR1 یا OCR1x) و حداکثر آن ۱٦ بیت می باشد (با دادن مقدار 0xFFFF به رجیسترهای ICR1یا OCR1x).

دقت موج PWM بر حسب مقدار ماکزیمم از رابطه ی زیر به دست می آید:

$$resolution = \frac{\log(TOP + ۱)}{\log(۲)}$$

پرچم سرریز تایمر TOV1 با رسیدن تایمر به مقدار صفر یک خواهد شد و با تطبیق مقایسه نیز بیت OCF1A یا OCF1B یک می شود. در این حالت ها اگر وقفه ی مربوطه فعال شده باشد برنامه می تواند به ISR آن وقفه منشعب شود. مقدار مقایسه (OCRx) را در ISR یا هر زمان دیگر می توان تغییر داد اما این مقدار در بافر رجیسترهای OCR1A و OCR1B ذخیره شده و با رسیدن تایمر به مقدار TOP در خود رجیستر Load می شود بنابراین تغییر دادن مقدار رجیسترهای OCR1x به دلیل تغییر آن با رسیدن به TOP می تواند باعث خروجی PWM نا متقارن شود:



✓ مشکل بالا در PWM تصحیح فاز و فرکانس با بروز کردن رجیسترهای OCR1x در زمان رسیدن به صفر، حل می شود.

تغییر وضعیت پین های OC1A و OC1B در حالت تطبیق مقایسه و سرریز مطابق جدول زیر خواهد بود:

COM1A1 /COM1B1	COM1A0 /COM1B0	وضعیت پین OC1A یا OC1B
0	0	غیر فعال (I/O معمولی)
0	1	اگر WGM1[3:0] = 9, 14 باشد: Toggle پین OC1A در وضعیت تطابق و OC1B پین I/O معمولی برای دیگر حالت های WGM1[3:0]: غیر فعال (I/O معمولی)
1	0	Clear در وضعیت تطابق و شمارش صعودی. Set در وضعیت تطابق و شمارش نزولی
1	1	Set در وضعیت تطابق و شمارش صعودی. Clear در وضعیت تطابق و شمارش نزولی

فرکانس موج PWM در حالت تصحیح فاز نصف حالت Fast PWM بوده و از رابطه ی زیر بدست می آید:

$$f_{PWM} = \frac{f_{Clk_I/O}}{۲.N.(۱+TOP)}$$

مثال ۱۰: در برنامه ی مثال قبل Mode تایمر را از Fast PWM به Phase Correct PWM تغییر داده و نصف شدن فرکانس PWM خروجی را مشاهده کنید:

```
TCCR1A=0x83;
TCCR1B=0x02;
OCR1AL=0xFF;
```

⑤ Phase and Frequency Correct Mode

همانطور که گفته شد به دلیل بروز کردن رجیستر OCR1x با رسیدن به TOP ممکن است شکل موج خروجی نامتقارن شود بنابراین برای حل این مشکل Mode پنجم تایمر یک این رجیستر را با رسیدن به صفر بروز می کند. تفاوت دیگر این Mode و عملکرد قبلی در این است که تایمر تنها در دو حالت زیر کار می کند: (۸، ۹ = WGM1[3:0])

۱. PWM تصحیح فاز و فرکانس با ICR1 = TOP
۲. PWM تصحیح فاز و فرکانس با OCR1A = TOP

واحد Capture تایمر یک

عملکرد این واحد به این صورت است که در اثر تریگر شدن ورودی Capture از پین ICP1 یا خروجی مقایسه گر آنالوگ مقدار موجود در رجیستر TCNT1 در رجیستر ICR1 نوشته شده و همزمان پرچم Capture تـایمر یک (ICF1) یک می شود. در این زمان در صورت فعال بودن بیـت پـرچم ورودی Capture (TICIE1) ایـن تریگر شدن می تواند باعث ایجاد وقفه شود. با اجرا شدن ISR به طور خودکار بیـت ICF1 صـفر شـده و یـا در صورت فعال نبودن وقفه می تواند با نوشتن یک بر روی آن پاک شود.

Bit	7	6	5	4	3	2	1	0
ICR1H	ICR1[15:8]							
ICR1L	ICR1[7:0]							

- رجیستر ICR1 به جز در حالتی که به عنوان TOP جهت مقایسه به کار می رود (Mode های ۸، ۱۰، ۱۲ و ١٤) یک رجیستر فقط خواندنی است.

همانطور که گفته شد تریگر شدن واحد Capture می تواند از دو منبع مختلف صورت گیرد که این از طریق بیت ACIC در رجیستر ACSR صورت می گیرد. صفر بودن این بیت پین ICP1 و یک بـودن آن خروجـی مقایسـه کننده ی آنالوگ را انتخاب می کند. همچنین نوع سیگنال ورودی از پین ICP1 بوسیله بیـت ICES1 از رجیسـتر TCCR1B تعیین می شود، به این ترتیب که صفر بودن این بیت لبه ی پایین رونده و یـک بـودن آن لبـه ی بـالا رونده ی سیگنال ورودی را انتخاب می کند.

TCCR1B	7	6	5	4	3	2	1	0
نام بیت	**ICNC1**	**ICES1**	-	WGM13	WGM12	CS12	CS11	CS10

ورودی Capture دارای یک واحد کاهش نویز نیز می باشد که با استفاده از یک فیلتر دیجیتـال ایمنـی ورودی را بهبود می بخشد. این واحد با یک کردن بیت ICNC1 از رجیستر TCCR1B فعال می شود. با فعـال شـدن ایـن فیلتر باید سیگنال نمونه برداری شده روی پایه ی ICP1 برای چهار سیکل کلاک معتبر باشد.

پروژه ۱۰: کنترل سروُ موتور

```
/*****************************************************
Project : Servo Motor Controller
Author  : Reza Sepas Yar
Company : Pishro Noavaran Kavosh
Chip type           : ATmega16
Clock frequency     : 16.000000 MHz
*****************************************************/

#include <mega16.h>
#include <delay.h>
#define xtal 16000000

void main(void)
{

PORTD=0x00;
```

```
DDRD=0x20;

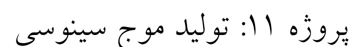
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 2000.000 kHz
// Mode: Ph. & fr. cor. PWM top=ICR1
// OC1A output: Non-Inv.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x80;
TCCR1B=0x12;
ICR1H=0x4E;
ICR1L=0x20;  //ICR=20000
OCR1AH=0x03;
OCR1AL=0xE8; //1000

while (1)
    {

    for(OCR1A=1000;OCR1A<2000;OCR1A++)
        delay_ms(1);

    for(OCR1A=2000;OCR1A>1000;OCR1A--)
        delay_ms(1);

    };
}
```




```
#include <mega16.h>
#define xtal 8000000

flash char sinewave[256]={
0x80,0x83,0x86,0x89,0x8C,0x8F,0x92,0x95,0x99,0x9C,0x9F,0xA2,0xA5,0xA8,0xAB,0xAE,
0xB1,0xB4,0xB6,0xB9,0xBC,0xBF,0xC1,0xC4,0xC7,0xC9,0xCC,0xCE,0xD1,0xD3,0xD6,0xD8,
0xDA,0xDC,0xDF,0xE1,0xE3,0xE5,0xE6,0xE8,0xEA,0xEC,0xED,0xEF,0xF1,0xF2,0xF3,0xF5,
0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFC,0xFD,0xFE,0xFE,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFE,0xFE,0xFE,0xFD,0xFC,0xFC,0xFB,0xFA,0xF9,0xF8,0xF7,
0xF5,0xF4,0xF3,0xF2,0xF0,0xEF,0xED,0xEB,0xEA,0xE8,0xE6,0xE4,0xE2,0xE0,0xDE,0xDC,
0xD9,0xD7,0xD5,0xD2,0xD0,0xCE,0xCB,0xC8,0xC6,0xC3,0xC1,0xBE,0xBB,0xB8,0xB5,0xB3,
0xB0,0xAD,0xAA,0xA7,0xA4,0xA1,0x9E,0x9B,0x97,0x94,0x91,0x8E,0x8B,0x88,0x85,0x82,
0x7E,0x7B,0x78,0x75,0x72,0x6F,0x6C,0x69,0x65,0x62,0x5F,0x5C,0x59,0x56,0x53,0x50,
0x4D,0x4B,0x48,0x45,0x42,0x3F,0x3D,0x3A,0x37,0x35,0x32,0x30,0x2D,0x2B,0x29,0x26,
0x24,0x22,0x20,0x1E,0x1C,0x1A,0x18,0x16,0x14,0x13,0x11,0x0F,0x0E,0x0D,0x0B,0x0A,
0x09,0x08,0x06,0x05,0x05,0x04,0x03,0x02,0x02,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
```

```
0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x02,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,
0x0A,0x0B,0x0D,0x0E,0x0F,0x11,0x13,0x14,0x16,0x18,0x1A,0x1C,0x1E,0x20,0x22,0x24,
0x26,0x29,0x2B,0x2D,0x30,0x32,0x35,0x37,0x3A,0x3D,0x3F,0x42,0x45,0x48,0x4B,0x4D,
0x50,0x53,0x56,0x59,0x5C,0x5F,0x62,0x65,0x69,0x6C,0x6F,0x72,0x75,0x78,0x7B,0x7E
};

char i=0;

interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
   OCR1A=sinewave[i];
   i++;
   if(i==255)
   i=0;
}

void main(void) {

DDRD=0xFF;

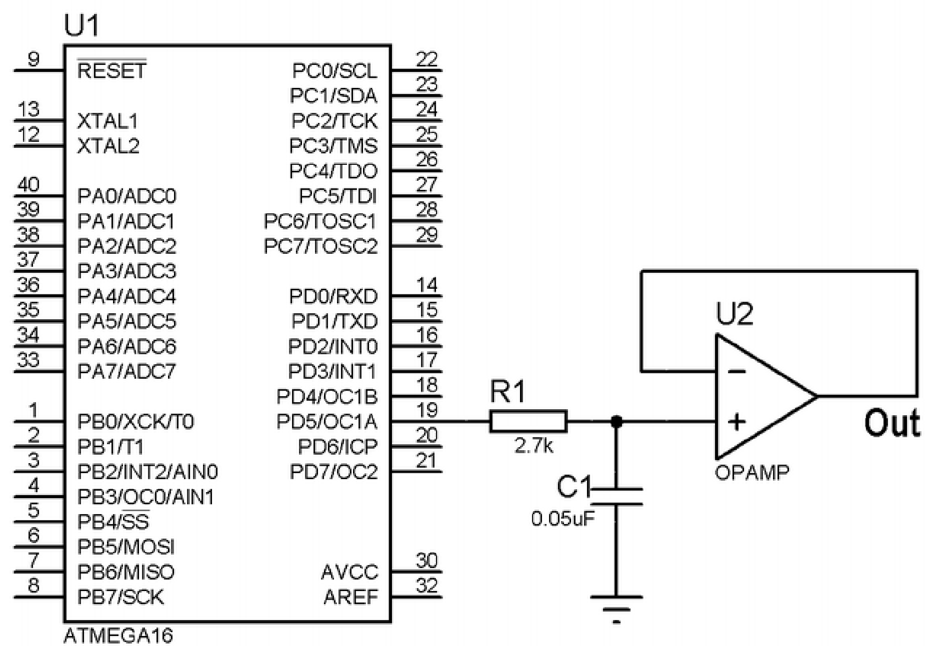
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 8000.000 kHz
// Mode: Fast PWM top=00FFh
// OC1A output: Non-Inv.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x81;
TCCR1B=0x09;

// Timer(s)/Counter(s) Interrupt(s) initialization
```

```
TIMSK=0x10;


//enable global interrups
#asm("sei");


    while (1);
}
```

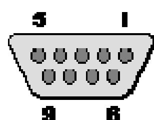


پورت سریال (RS-232)

RS-232c در اواخر دهه ٦٠ میلادی به صورت استاندارد تعریف شد و همچنان یکی از استانداردهای پرکاربرد در کامپیوترهای شخصی و کاربردهای صنعتی است. این استاندارد هم ارتباط سریال سنکرون و هم آسنکرون را پشتیبانی کرده و به صورت Full Duplex عمل می نماید. کامپیوترهای شخصی تنها ارتباط آسنکرون را پشتیبانی می کنند و از طریق چیپ UART موجود در برد اصلی، اطلاعات را از حالت موازی به سریال یا از سریال به موازی تبدیل کرده و با تنظیمات زمانی آن را از طریق پورت سریال ارسال یا دریافت می کند.



پورت سریال دارای یک کانکتور ٩ پین می باشد و از آنجایی که این استاندارد در ابتدا برای ارتباط با مودم طراحی شده بود، دارای پین های Handshaking و وضعیت می باشد. اما نوع خاصی از ارتباط با RS-232 به نام Null-Modem که تنها شامل پین های ارسال و دریافت است برای ارتباط با غیر از مودم استفاده می شود. بنابراین تنها دو پین Rx و Tx (و البته زمین) مورد نیاز است. در شکل زیر کانکتور پورت سریال را که D9 نام دارد ملاحظه می کنید:

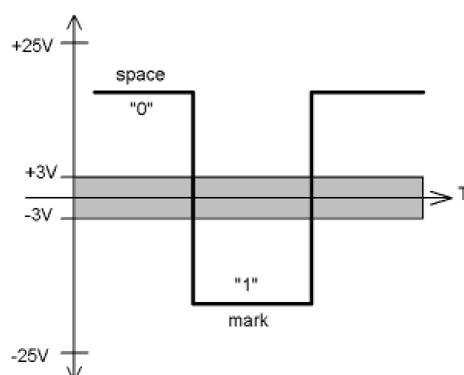


در جدول زیر عملکرد هر پین آورده شده است:

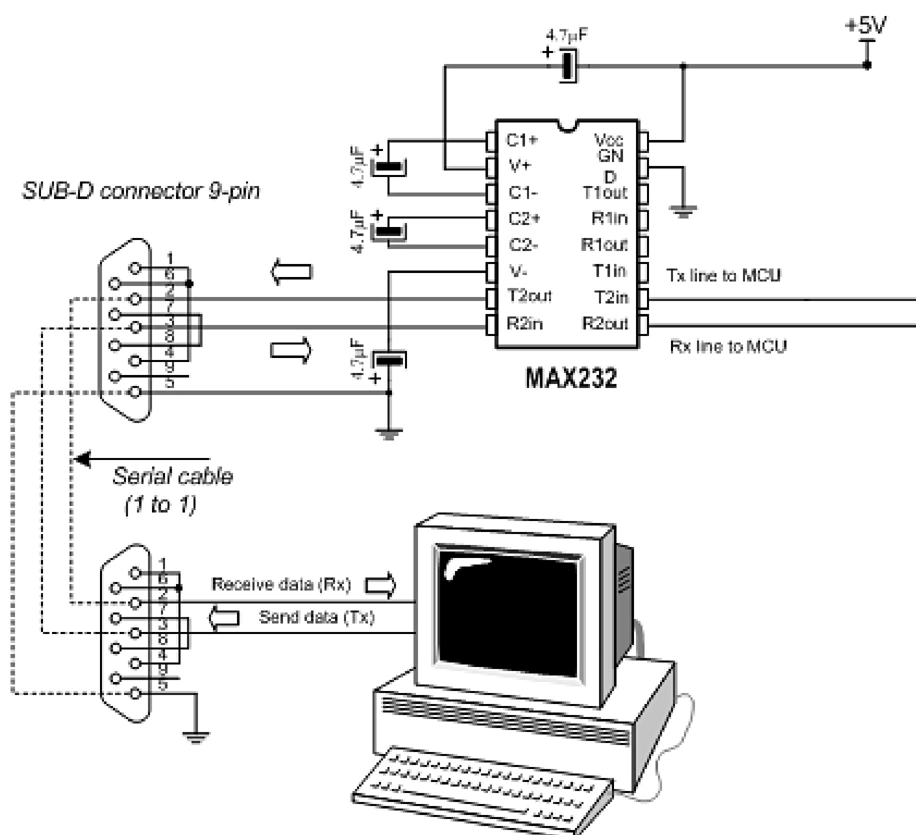
Pin		عملکرد
۱	Carrier Detect	آیا مودم به یک خط تلفن متصل است ؟
۲	Receive Data	کامپیوتر اطلاعات ارسال شده توسط مودم را دریافت می نماید.
۳	Transmit Data	کامپیوتر اطلاعاتی را برای مودم ارسال می دارد.
٤	Data Terminal Ready	کامپیوتر به مودم آمادگی خود را برای ارتباط اعلام می دارد.
٥	Signal Ground	زمین سیگنال
٦	Data Set Ready	مودم آمادگی خود را برای ارتباط به کامپیوتر اعلام می دارد.
٧	Request To Send	کامپیوتر از مودم در رابطه با ارسال اطلاعات سوال می نماید.
٨	Clear To Send	مودم به کامپیوتر اعلام می نماید که می تواند اطلاعاتی را ارسال دارد.
٩	Ring Indicator	زنگ تلفن تشخیص داده خواهد شد.

سطح سیگنال:

در استاندارد RS-232 سطح ولتاژ ۳+ تا ۱۲+ نمایانگر وضعیت Space یا صفر منطقی و بازه ی ۳- تا ۱۲- ولت نمایشگر وضعیت Mark یا یک منطقی می باشد:

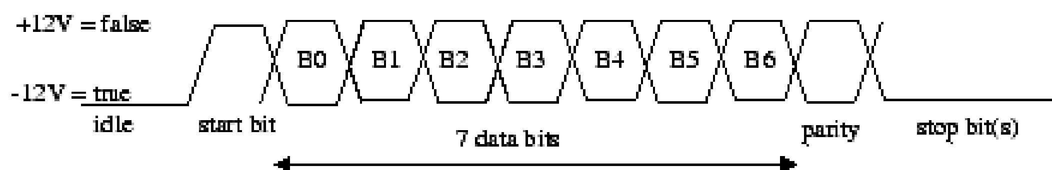


اگرچه تجهیزات استاندارد TTL با سطوح منطقی ۰ و ۵ ولت کار می کنند اما قالب اطلاعات ارسالی تفاوتی ندارد و با یک مدار تغییر سطح ولتاژ، PC می تواند با ادوات TTL ارتباط برقرار نماید. یکی از مبدل های متداول سطح RS-232 به TTL مدار مجتمع MAX232 و یا HIN232 می باشد. MAX232 یک تراشه ی ۱۶ پایه است که شامل ۲ فرستنده و ۲ مبدل مجزا است. در زیر یک مدار نمونه را برای کار با این IC مشاهده می کنید:



قالب اطلاعات:

در یک Frame اطلاعاتی که توسط بیت شروع و بیت پایان محصور شده است معمولا ۵ تا ۸ بیت دیتا قرار می گیرد و یک بیت توازن نیز به صورت اختیاری تعریف می شود. بیت شروع متناظر با صفر منطقی است و بیت پایان (که ممکن است ۱ یا ۲ بیت باشد.) توسط یک شناسایی می شود. مثلا در نمودار زمانی زیر یک Frame شامل ۱۰ بیت است که هفت بیت آن شامل Data یک بیت آغازین و یک بیت پایانی و یک بیت توازن قبل از بیت پایان می باشد:



عملکرد USART میکروکنترلر AVR

قطعه ی ATmega16 دارای یک ماژول USART بوده که از استاندارد RS-232 در دو حالت آسنکرون و سنکرون پشتیبانی می کند. دسترسی به پورت سریال AVR از طریق سه پین TXD، RXD و XCK که به ترتیب پین ارسال، دریافت وکلاک می باشند امکان پذیر است. (پین XCK فقط در Mode سنکرون کاربرد دارد.)

قالب اطلاعات در USART میکروکنترلر AVR همانند UART کامپیوترهای شخصی شامل یک بیت شروع، بیت های داده، بیت اختیاری توازن و یک یا دو بیت پایان است، با این تفاوت که در AVR می تواند ۵ تا ۹ بیت

Data تعریف شود. بیت تـوازن مـی توانـد فـرد یـا زوج باشـد و از طریـق بیـت هـای UPM[1:0] از رجیسـتر UCSRC تنظیم می شود.

رجیسترهای USART

۱. (UDR) USART Data Register

Bit	7	6	5	4	3	2	1	0
UDR (Read)	**RXB[7:0]**							
UDR (Write)	**TXB[7:0]**							

بافر دریافت و ارسال پورت سریال دارای یک آدرس مشترک به نام UDR در فضای I/O Registers می باشند. بافر ارسال، مقصد داده های نوشته شده در رجیسترUDR بوده و خواندن این رجیستر محتویـات بـافر دریافـت را به دست می دهد. تنها زمانی مـی تـوان روی رجیسـتر UDR مقـداری را نوشـت کـه بیـت UDRE از رجیسـتر UCSRA یک شده باشد و در غیر اینصورت دیتای ارسالی توسط USART نادیده گرفته مـی شـود. بـا ارسـال اطلاعات به بافر ارسال USART در صورتی که بیـت TXEN از رجیسـتر UCSRB یـک باشـد اطلاعـات در شیفت رجیستر بارگذاری شده و بیت به بیت از پین TXD ارسال می شود.

۲. USART Control and Status Register A

UCSRA	7	6	5	4	3	2	1	0
نام بیت	RX	TXC	UDR	FE	DOR	PE	U2	MPC

	C		E				X	M

Multi-processor Communication Mode: یک شدن این بیت میکروکنترلر را به حالت ارتباطات چند پردازنده ای می برد.

Double the USART Transmission Speed: با یک کردن این بیت در Mode آسنکرون Baud Rate دو برابر خواهد شد. در Mode سنکرون این بیت باید صفر باشد.

Parity Error: در صورت فعال بودن تولید بیت توازن از طریق بیت های UPM[1:0]، با روی دادن خطای توازن در بافر دریافت، این بیت یک می شود.

Data Overrun: با بروز Overrun این بیت یک می شود. شرایط Overrun یا لبریز وقتی روی می دهد که بافر دریافت پر باشد و شیفت رجیستر نیز محتوی داده ی جدیدی باشد و داده ی جدیدی نیز از راه برسد، یعنی یک بایت در بافر شیفت رجیستر منتظر باشد و بیت شروع جدیدی دریافت شود. در این حالت اطلاعات جدید از بین می رود و با یک شدن بیت DOR بروز خطا اعلام می شود.

Frame Error: اگر در Frame دریافت شده بیت پایان صفر باشد این بیت یک شده و در غیر اینصورت صفر خواهد بود.

USART Data Register Empty: یک بودن این پرچم نشان دهنده ی این است که اطلاعات موجود در بافر ارسال برای شیفت رجیستر ارسال شده و بافر ارسال آماده ی دریافت کاراکتر جدید است. همچنین در صورتی که بیت UDRIE از رجیستر UCSRB یک باشد باعث ایجاد وقفه شده و تا زمانی که بیت UDRE یک است با خارج شدن از ISR دوباره آن را اجرا می کند **با نوشتن داده ی جدید در UDR پرچم UDRE صفر می شود**. بعد از ریست شدن میکرو این بیت یک می شود که به معنای آماده بودن دریافت کاراکتر جدید است.

USART Transmit Complete: این پرچم زمانی یک می شود که تمام اطلاعات موجود در شیفت رجیستر به بیرون شیفت داده شده و داده ی جدیدی در بافر ارسال وجود نداشته باشد. با فعال بودن بیت TXCIE این پرچم می تواند باعث ایجاد وقفه ی کامل شدن ارسال شود و با **اجرای ISR بیت TXC توسط سخت افزار پاک شده و در غیر اینصورت نرم افزار می تواند با نوشتن یک بر روی آن، آن را پاک کند.**

USART Receive Complete: این بیت با کامل شدن دریافت یک Frame در UDR یک شده و **پس از خواندن UDR صفر می شود.**

۳. USART Control and Status Register B

UCSRB	7	6	5	4	3	2	1	0
نام بیت	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

Transmit Data Bit 8: در حالتی که از پورت سریال در Mode ۹ بیتی استفاده می شود این بیت نهمین بیت کاراکتر ارسالی خواهد بود. باید توجه داشت که قبل از نوشتن در UDR باید وضعیت این بیت را مشخص کرد.

Receive Data Bit 8: در حالتی که از پورت سریال در Mode ۹ بیتی استفاده می شود این بیت نهمین بیت کاراکتر دریافتی خواهد بود. باید توجه داشت که قبل از خواندن UDR باید این بیت را خواند.

Character Size: با ترکیب این بیت و بیت های UCSZ[1:0] در رجیستر UCSRC تعداد بیت های داده (اندازه ی کاراکتر) را در یک Frame مشخص می کند.

Transmitter Enable: با یک کردن این بیت عملکرد عادی پین TxD به ارسال پورت سـریال تغییـر حالـت داده و بعد از آن قابل استفاده به صورت I/O معمولی نیست. غیر فعال کردن این بیت در حالیکه UART سـریال مشغول است تا اتمام ارسال اطلاعات تاثیر نخواهد گرفت.

Receiver Enable: با یک کردن این بیت عملکرد عادی پین RxD به دریافت پورت سریال تغییر حالـت داده و بعد از آن قابل استفاده به صورت I/O معمولی نیست. با صفر کردن این بیت بافر پـورت سـریال خـالی شـده و مقدار بیت های DOR، FE و PE نامعتبر خواهد بود.

USART Data Register Empty Interrupt Enable: با یک شدن این بیـت، در صـورتی کـه بیـت فعال ساز کلی وقفه ها (I) یک باشد، فعال شدن پرچم خالی بودن بافر ارسال (UDRE) مـی توانـد باعـث ایجـاد وقفه شود.

TX Complete Interrupt Enable: با یک شدن این بیت، در صورتی که بیت فعال ساز کلی وقفه هـا (I) یک باشد، فعال شدن پرچم اتمام ارسال (TXC) می تواند باعث ایجاد وقفه شود.

RX Complete Interrupt Enable: با یک شدن این بیت، در صورتی که بیت فعال ساز کلی وقفه هـا (I) یک باشد، فعال شدن پرچم اتمام ارسال (RXC) می تواند باعث ایجاد وقفه شود.

٤. USART Control and Status Register C

UCSRC	7	6	5	4	3	2	1	0
نام بیت	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

رجیسترهای UCSRC و UBRRH دارای آدرس مشترکی در فضای I/O هستند، که برای انتخاب این دو از بیت URSEL استفاده می شود.

Clock Polarity: این بیت فقط در Mode سنکرون به کار برده می شود و در Mode آسنکرون باید روی آن صفر نوشته شود. عملکرد این بیت مطابق جدول زیر است:

UCPOL	تغییر داده ی ارسالی روی پین TxD	نمونه برداری از داده ی دریافتی روی پین RxD
۰	لبه ی بالا رونده ی پالس XCK	لبه ی پایین رونده ی پالس XCK
۱	لبه ی پایین رونده ی پالس XCK	لبه ی بالا رونده ی پالس XCK

Character Size[1:0]: این بیت به همراه بیت UCSZ2 مطابق جدول زیر تعداد بیت های یک کاراکتر را تعیین می کنند، در حالت پیش فرض این بیت ها اندازه ی ۸ بیت را تعیین می کنند.

UCSZ2	UCSZ1	UCSZ0	اندازه ی کاراکتر
۰	۰	۰	۵ بیت
۰	۰	۱	٦ بیت
۰	۱	۰	۷ بیت
۰	۱	۱	۸ بیت
۱	۰	۰	رزرو شده

رزرو شده	۱	۰	۱
رزرو شده	۰	۱	۱
حالت ۹ بیتی	۱	۱	۱

Stop Bit Select: این بیت، تعداد بیت های پایان را معین می کند. در حالت پیش فرض که این بیت صفر می باشد یک بیت پایان و در صورت ۱ شدن ۲ بیت پایان در نظر گرفته می شود.

Parity Mode[1:0]: این دو بیت تنظیمات مربوط به بیت توازن را مطابق جدول زیر انجام می دهند. در صورت فعال بودن بیت توازن همراه هر Frame این بیت ایجاد شده و در گیرنده بر حسب همان تنظیمات (که باید در آنجا نیز مطابق تنظیمات فرستنده انجام شود.) مقدار بیت توازن با مقدار دریافت شده مقایسه شده و در صورت بروز خطا پرچم PE از رجیستر UCSRA یک می شود.

وضعیت بیت توازن	UPM0	UPM1
غیر فعال	۰	۰
رزرو شده	۱	۰
توازن زوج	۰	۱
توازن فرد	۱	۱

USART Mode Select: UMSEL حالت کار UART را تعیین می کند. در صورتی این بیت یک باشد در حالت آسنکرون و با یک کردن آن در وضعیت سنکرون کار می کند.

Register Select: همانطور که گفته شد دو رجیستر UCSRC و UBRRH دارای آدرس مشترکی (0x40) هستند و این بیت برای انتخاب نوشتن روی یکی از این دو رجیستر می باشد. به این صورت که اگر بخواهیم روی UCSRC مقداری را بنویسیم باید در همان بایت MSB یک باشد و چنانچه بخواهیم روی UBRRH مقداری را بنویسیم باید MSB صفر باشد. به عنوان مثال دستورات زیر مقدار بیت ۱ از هر رجیستر را یک می کند:

```
UBRRH = 0x02;
UCSRC = 0x82;
```

روش خواندن این دو رجیستر به این صورت است که در صورت خوانـدن مقـدار آدرس 0x40 و یـا هـر یـک از اسامی مستعار این آدرس (UCSRC یا UBRRH) مقدار رجیستر UBRRH به دست خواهد آمد:

```
a = UBRRH;
```

چنانچه بخواهیم مقدار رجیستر UCSRC را بدست آوریم باید در سیکل کلاک بعدی مقدار خوانـده شـده را بـه عنوان مقدار واقعی UCSRC در نظر بگیریم، به عنوان مثال:

```
b = UBRRH;
b = UCSRC;
```

- ✓ بهتر است در صورت اسـتفاده از وقفـه، قبـل از پروسـه ی بـالا بیـت فعـال سـاز وقفـه هـا را بـا دسـتور `#asm("cli")` غیر فعال کنیم.
- ✓ با خواندن UBRRH بیت URSEL صفر خوانده شده و با خواندن UCSRC این بیت یک خوانده می شود.

۵. USART Baud Rate Registers

Bit	7	6	5	4	3	2	1	0
UBRRH	URSEL	-	-	-	UBRR[11:8]			
UBRRL	UBRR[7:0]							

URSEL: همان طور که گفته شد برای انتخاب بین UBRRH و UCSRC استفاده می شود.

UBRRH[7:4]: برای کاربردهای آینده رزرو شده هستند و برای سازگاری با قطعات آتی باید روی آن ها صفر نوشته شود.

UBRR[11:0]: این دوازده بیت برای تعیین Baud Rate رابط USART مورد استفاده قرار می گیرند. به این منظور می توان از روابط زیر استفاده نمود:

Mode کاری	محاسبه Baud Rate از روی UBRR	محاسبه UBRR از روی Baud Rate
آسنکرون (U2X=0)	$Baud = \frac{f_{osc}}{16(UBRR+1)}$	$UBRR = \frac{f_{osc}}{16 \times Baud} - 1$
آسنکرون با سرعت دوبرابر (U2X=1)	$Baud = \frac{f_{osc}}{8(UBRR+1)}$	$UBRR = \frac{f_{osc}}{8 \times Baud} - 1$
عملکرد سنکرون	$Baud = \frac{f_{osc}}{2(UBRR+1)}$	$UBRR = \frac{f_{osc}}{2 \times Baud} - 1$

مقدار خطای بوجود آمده از رابطه ی زیر بدست می آید:

$$Error[\%] = (\frac{BaudRate_{ClosestMatch}}{BaudRate} - 1) \times 100\%$$

مثال ۱: با کلاک ۸ مگاهرتز و Baud Rate = ۲٤۰۰ در Mode آسنکرون عادی، مقدار UBRR را بدست آورید.

$$UBRR = \frac{۸۰۰۰۰۰۰}{۱٦ \times ۲٤۰۰} - ۱ = ۲۰۷.۳۳ \Rightarrow UBRR = ۲۰۷$$

$$BaudRate = \frac{۸۰۰۰۰۰۰}{۱٦(۲۰۷ + ۱)} = ۲٤۰٤$$

$$Error[\%] = (\frac{۲٤۰٤}{۲٤۰۰} - ۱) \times ۱۰۰\% = ۰.۲\%$$

برای صفر شدن خطا می توان از کرسیتال 7.3728 مگاهرتز استفاده نمود اگرچه حداکثر خطای قابل قبول برای این وضعیت ۲ درصد می باشد.

- ✓ کریستال هایی که می توان با استفاده از آن ها به خطای Baud Rate صفر رسید ۱.۸٤۳۲، ۳.٦۸٦٤، ۷.۳۷۲۸، ۱۱.۰۵۹۲، ۱٤.۷٤٥٦ می باشند.
- ✓ مقدار UBRR و خطاهای حاصل در کلیه حالت ها در صفحات ۱٦۹– ۱٦٦ از برگه ی اطلاعاتی ATmega16 موجود می باشد.

توابع USART در CodeVision

اعلان این توابع در فایل **stdio.h** قرار دارد و قبل از استفاده از آن ها باید تنظیمات اولیه USART انجام شود.

تابع getchar(): این تابع به روش Polling منتظر می ماند تا پرچم RXC یک شده و بعد از آن مقدار UDR را می خواند.

تابع ()putchar: این تابع به روش Polling منتظر می ماند تـا پـرچم UDRE یـک شـده و بعـد از آن یـک کاراکتر را در UDR کپی می کند.

مثال۲: (میکرو اول بعد از ۳ ثانیه عدد ۷ را برای میکرو دوم ارسال می کند و در مدت این سه ثانیه میکـرو دوم بـه روش Polling منتظر دریافت یک کاراکتر می ماند و پس از دریافت آن را در PORTA می ریزد.)

برنامه ی میکرو اول:

```
#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#define xtal 8000000


void main(void)
{


UCSRA=0x00;
UCSRB=0x08; // USART Transmitter: On
UCSRC=0x86; //8 Data, 1 Stop, No Parity
UBRRH=0x00;
UBRRL=0x33; // USART Baud rate: 9600


      delay_ms(3000);
      putchar(7);
      while (1);
}
```

برنامه میکرو دوم:

```
#include <mega16.h>
#include <stdio.h>
```

```
#define xtal 8000000


void main(void)
{


char a;
DDRA=0xFF;
PORTA=0xFF;


UCSRA=0x00;
UCSRB=0x10; // USART Receiver: On
UCSRC=0x86; //8 Data, 1 Stop, No Parity
UBRRH=0x00;
UBRRL=0x33; // USART Baud rate: 9600

      a = getchar(); // Polling RXC
      PORTA = a;
      while (1);


}
```

توابع سطح بالای USART تنها برای صرفه جویی در وقت برنامه نویس می باشند. در صورت نیاز می توان به صورت مستقیم با رجیسترها کار کرد. به عنوان مثال حلقه ی اصلی برنامه ی میکرو دوم را به صورت زیر نیز می توان نوشت:

```
      while(!(UCSRA&0x80));
      PORTA=UDR;
      while (1);
```

توابع ()puts و ()putsf: این توابع یک رشته را توسط USART به پورت سریال ارسال می کنند. تابع ()puts رشته ای را که در SRAM است و تابع ()putsf رشته ای را که در Flash است به خروجی ارسال می کند. اساس تمام توابع سطح بالای USART توابع ()putchar و ()getchar می باشند و در اینجا نیز با استفاده از اشاره گر Y، کاراکترها پشت سر هم بوسیله ی این توابع به رجیستر UDR ارسال می شوند.

✓ این توابع به انتهای رشته کاراکتر LF (0x10) را اضافه می کنند.

مثال۳: (رشته های "Kavosh" و "AVR" به پورت سریال ارسال می شوند.)

```
#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#define xtal 8000000

flash char string1[7]="Kavosh";
char string2[7]="AVR";

void main(void)
{

UCSRA=0x00;
UCSRB=0x08; // USART Transmitter: On
UCSRC=0x86; //8 Data, 1 Stop, No Parity
```

```
UBRRH=0x00;
UBRRL=0x33; // USART Baud rate: 9600

      putsf(string1);
      puts(string2);
      while (1);
}
```

تابع ()gets : این تابع دارای دو آرگومان نام متغیر و طول است که منتظر می ماند تا کاراکترهای دریافتی به اندازه ی طول شده و سپس آن ها را داخل متغیر می ریزد.

مثال ٤: (میکرو اول بعد از ١ ثانیه عدد رشته ی Kavosh را برای میکرو دوم ارسال می کند و در مدت این یک ثانیه میکرو دوم به روش Polling منتظر دریافت رشته می ماند و پس از دریافت آن را در LCD نمایش می دهد.)

برنامه میکرو ١:

```
#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#define xtal 8000000

void main(void)
{

UCSRA=0x00;
UCSRB=0x08; // USART Transmitter: On
UCSRC=0x86; //8 Data, 1 Stop, No Parity
```

```
UBRRH=0x00;
UBRRL=0x33; // USART Baud rate: 9600


      delay_ms(1000);
      putsf(" Kavosh ");
      while (1);
}
```

برنامه میکرو ۲:

```
#include <mega16.h>
#include <stdio.h>
#include <lcd.h>
#define xtal 8000000


#asm
   .equ __lcd_port=0x1B ;PORTA
#endasm


char a[10];


void main(void)
{


UCSRA=0x00;
UCSRB=0x10; // USART Receiver: On
UCSRC=0x86; //8 Data, 1 Stop, No Parity
UBRRH=0x00;
```

```
UBRRL=0x33; // USART Baud rate: 9600


        lcd_init(16);
        lcd_clear();
      lcd_putsf("Waiting...");
        gets(a,10);
        lcd_clear();
      lcd_puts(a);
        while(1);


}
```

تابع ()printf: این تابع یک رشته ی قالب بندی شده را به پورت سریال ارسال می کند. کاراکتر فرمت می تواند مطابق جدول زیر باشد:

کاراکتر فرمت	نوع اطلاعات
%c	یک کاراکتر
%d و %i	عدد صحیح علامتدار در مبنای ۱۰
%e و %E	عدد اعشاری به صورت نماد علمی
%f	عدد اعشاری
%s	رشته ی ختم شده به Null واقع در SRAM
%p	رشته ی ختم شده به Null واقع در Flash

عدد صحیح بدون علامت در مبنای ۱۰	%u
عدد صحیح بدون علامت در مبنای ١٦	%x و %X
کاراکتر %	%%

مثال ٥: (یک رشته ی قالب بندی شده را به پورت سریال ارسال می کند.)

```
#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#define xtal 8000000

int a = 100;
char b = 'A';
float pi = 3.14;

void main(void)
{

UCSRA=0x00;
UCSRB=0x08; // USART Transmitter: On
UCSRC=0x86; //8 Data, 1 Stop, No Parity
UBRRH=0x00;
UBRRL=0x33; // USART Baud rate: 9600

      printf("a=%d b=%c pi=%f",a,b,pi);
```

```
        while (1);
}
```

تابع printf() این قابلیت را دارد که برنامه نویس طول میدان و دقت پارامتر را تعیین کند که این امکان به صورت %width.precision قابل تنظیم می باشد. width نشان دهنده ی طول یک عدد است که در مورد اعداد صحیح اگر بیشتر از تعداد ارقام باشد به اندازه ی اضافه جای خالی در سمت چپ عدد در نظر گرفته می شـود و چنانچـه عدد اعشاری باشد این مقدار بیانگر قسمت صحیح و "." و قسمت اعشاری مـی باشـد. precision مقـدار دقـت اعشار را تعیین می کند. به عنوان مثال اگر a = 3.145 باشد دستور زیر *a=3.14* را چاپ می کند.

```
printf("a=%4.2",a);
```

مشخصات تابع printf() در کامپایلر CodeVision را می توان از طریق:

Project/Configure/C Compiler/(s)printf Features

تعیین نمود.

تابع scanf(): با استفاده از این تابع می تواند قالب اطلاعات دریافتی از پورت سریال را تعیین نمود. طرز کار این تابع به این صورت است:

```
scanf("عبارت اول","عبارت دوم);
```

عبارت دوم نام متغیری است که رشته ی دریافتی در آن قرار می گیرد و عبارت اول تعیین می کند که قالب داده ی دریافتی به چه صورت باشد. به صورت پیش فرض طول رشته ی دریافتی ۲۵٦ کاراکتر است اما بـا افـزودن طـول قبل از کاراکتر فرمت می توان آن را به مقدار دلخواه کاهش داد. به عنوان مثال:

```
scanf("%10d",a)
```

۱۰ کاراکتر را از ورودی خوانده و در متغیر a ذخیره می کند.

مشخصات تابع `scanf()` را همانند `printf()` می توان از:

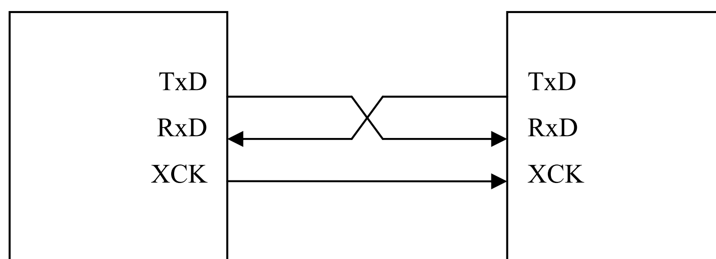
Project/Configure/C Compiler/(s)scanf Features

تنظیم نمود.

توجه: توابع printf() و scanf() حجم زیادی از حافظه ی Flash را مصرف کرده و سرعت اجرای آن ها پایین است، بنابراین تا حد امکان از آن ها استفاده نکرده و در صورت لزوم مشخصات آن ها را به ساده ترین حالت (int و int, width) تنظیم نمایید.

استفاده از پورت سریال در وضعیت سنکرون

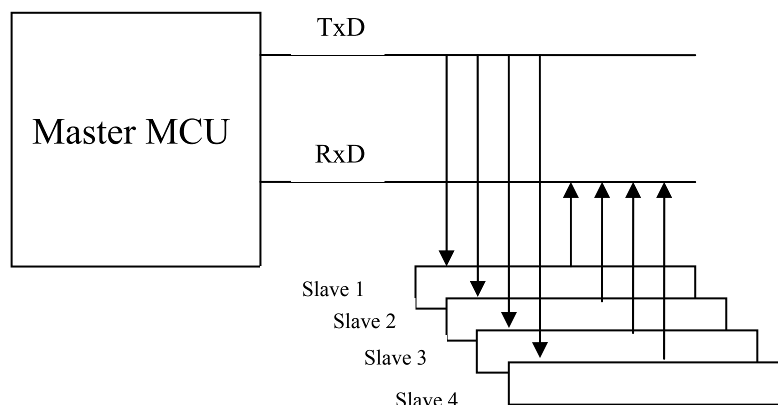
با یک کردن بیت UMSEL از رجیستر UCSRC میکروکنترلر در این حالت قرار می گیرد و اتصالات سخت افزاری به صورت زیر می باشد:



همانطور که گفته شد بوسیله ی بیت های UCPOL می توان لبه ی نمونه برداری و تغییر بر اسال کلاک را تعیین نمود. مسئله ای که باید در Mode سنکرون رعایت شود این است که داده باید در خلاف لبه ای که در فرستنده ارسال شده است، در گیرنده نمونه برداری شود یعنی اگر داده در لبه ی پایین رونده ارسال شده باشد در لبه ی بالا رونده دریافت شود.

وضعیت Multi-processor

برای ایجاد شبکه بین میکروکنترلرها از این Mode استفاده می شود. معمولا یک میکروکنترلر به عنوان Master و بقیه به عنوان Slave ایفای نقش می کنند.



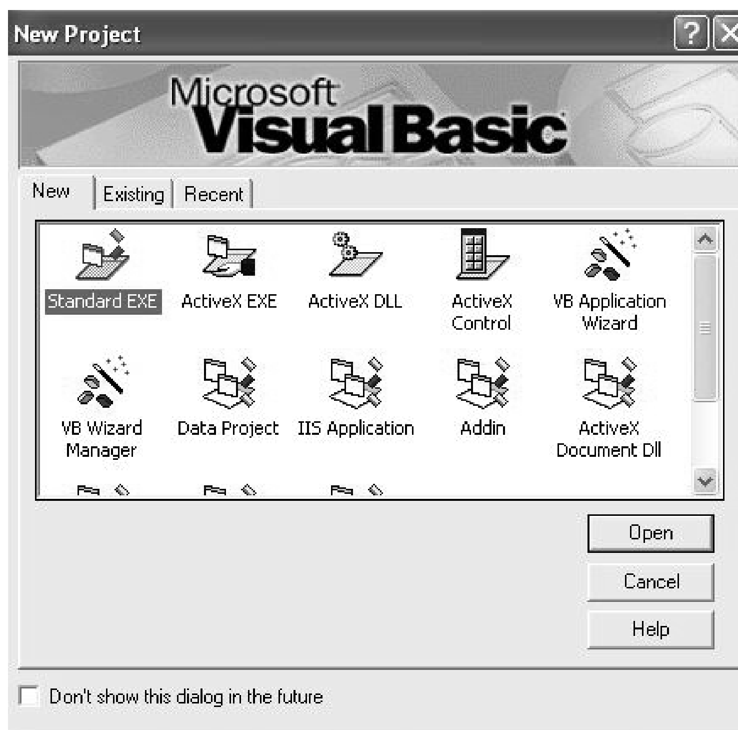
با یک کردن بیت MPCM از رجیستر UCSRA میکروکنترلر وارد این Mode خواهد شد. برای مبادله ی اطلاعات بین Master و Slave، باید Slave توسط Master آدرس دهی شود که بدین منظور از بین نهم داده (TXB8) استفاده می شود. آدرس Slave ها قبلا توسط برنامه نویس تعیین شده و تمام آن ها باید با یک شدن بیت MPCM در Mode چند پردازنده ای قرار بگیرند و منتظر دریافت آدرس از Master بمانند. برای شروع ارتباط Master یک فریم اطلاعاتی ۹ بیتی که هشت بیت آن شامل آدرس و بیت نهم آن یک است به میکروکنترلرها ارسال می کند. یک بودن بیت نهم باعث می شود که رفتار Slave ها با داده ی دریافتی همانند یک آدرس بوده و آن را با آدرس خود مقایسه کنند. Slave انتخاب شده بیت MPCM خود را پاک کرده و منتظر دریافت داده می ماند و بقیه Slave ها که بیت MPCM آن ها یک است همچنان منتظر دریافت آدرس می مانند. در این زمان Slave انتخاب شده با ارسال پیامی برای Master تصدیق می فرستند تا بیت نهم خود را صفر کند تا در ارسال بعدی به اشتباه Slave دیگری انتخاب نشود. پس از اتمام ارسال داده Master مجددا به نشانه ی آدرس دهی بیت نهم خود را یک کرده و آدرس دیگری را ارسال می کند. Slave قبل که بیت MPCM آن صفر شده بود مجددا این بیت را یک کرده و آماده ی دریافت آدرس می شود.

* پروژه ۱۲: پورت سریال در ویژوال بیسیک

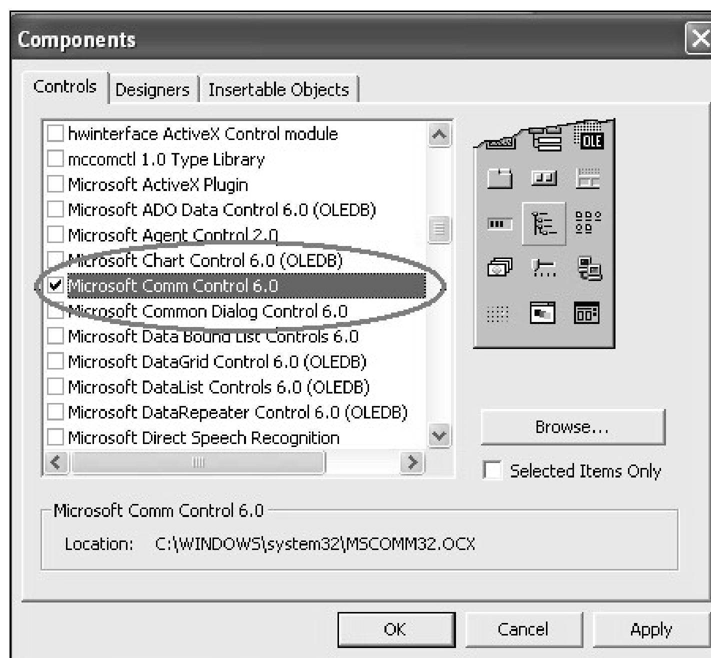
ارتباط بین کامپیوتر و PC به صورت Null-Modem بوده و همانطور که اشاره شد برای تغییر سطح RS-232 به TTL مطابق شماتیک ابتدای فصل از مبدل MAX232 استفاده می شود.

برای دسترسی به پورت های COM در ویژوال بیسیک می توان از کنترل MSComm که همراه کامپایلر وجود دارد استفاده نمود. این کنترل، ارتباط دهی پورت سریال را از طریق فایل mscomm32.ocx برای برنامه نویس فراهم می کند. برای آشنایی با این OCX مراحل زیر را انجام دهید:

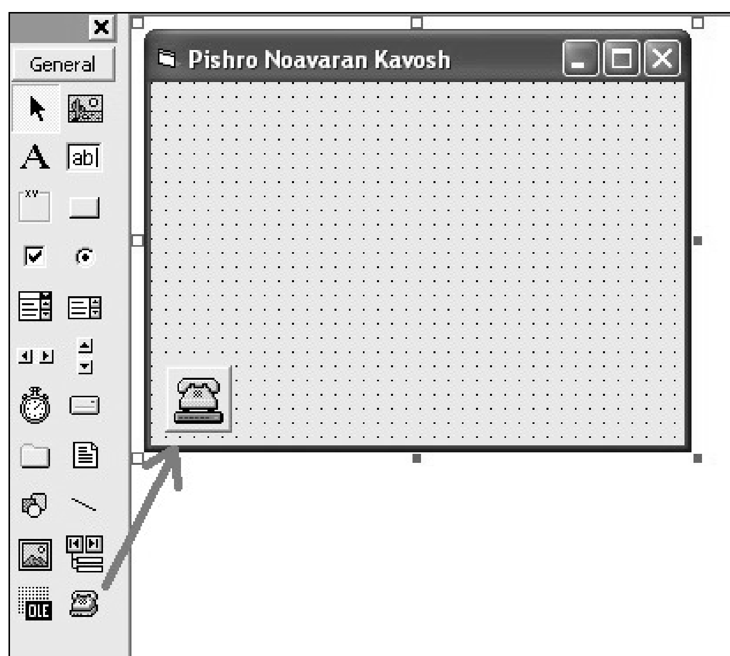
۱. یک پروژه ی جدید از نوع Standard EXE بسازید:



۲. از طریق منوی اصلی با کلیک بر روی گزینه ی Project مورد Component را انتخاب کنید و در کادر حاصل گزینه ی Microsoft Comm Control 6.0 را علامت زده و دکمه ی OK را کلیک کنید.



۳. آیکون کنترل MScomm را از Toolbox روی فرم قرار دهید:



۶. در رویداد Load فرم قطعه کد زیر را وارد کنید.

```
Private Sub Form_Load()

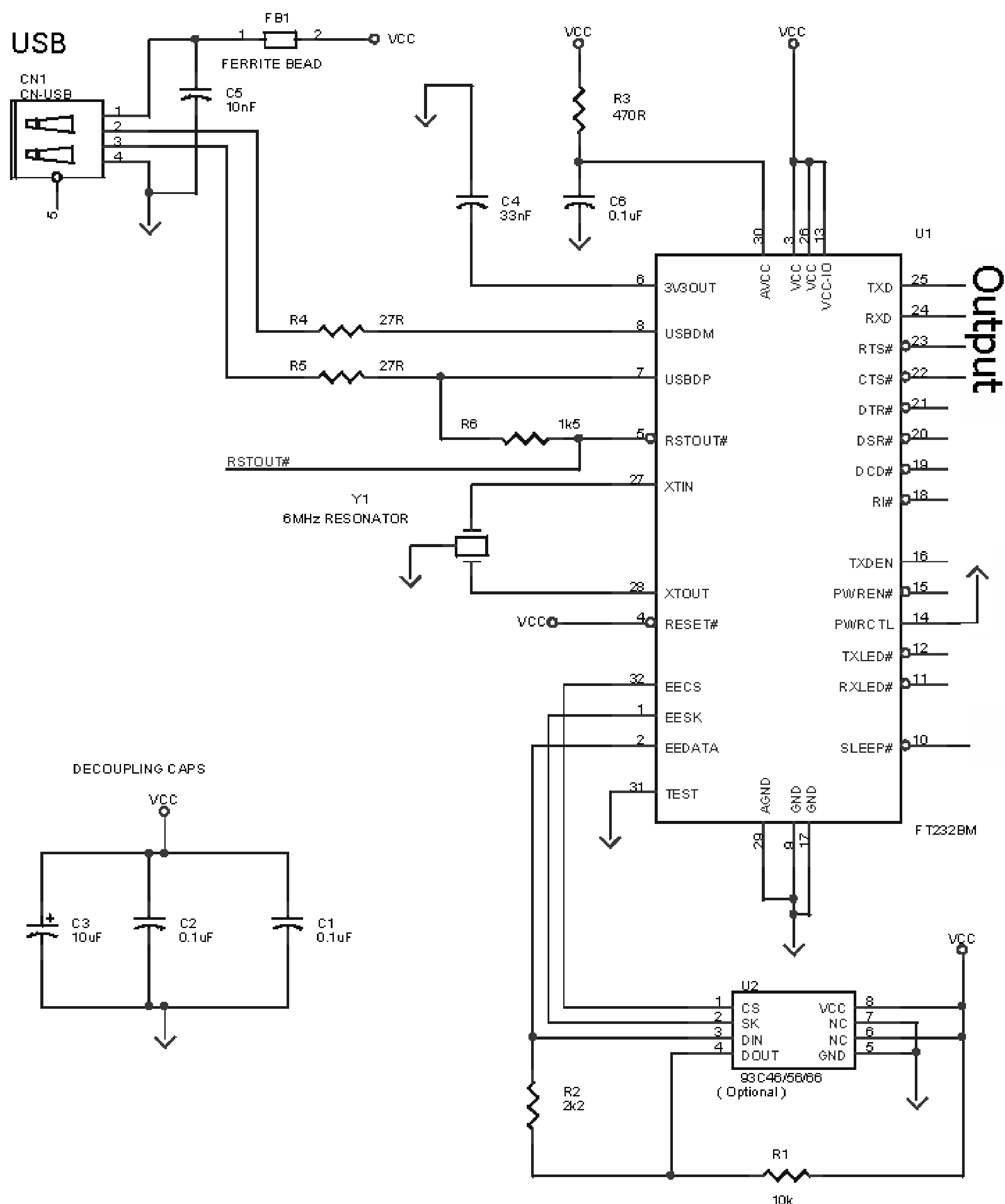
        MSComm1.Settings = "9600,N,8,1"
        MSComm1.CommPort = 1
        MSComm1.PortOpen = True
        MSComm1.Output = "Test" + Chr(16)

End Sub
```

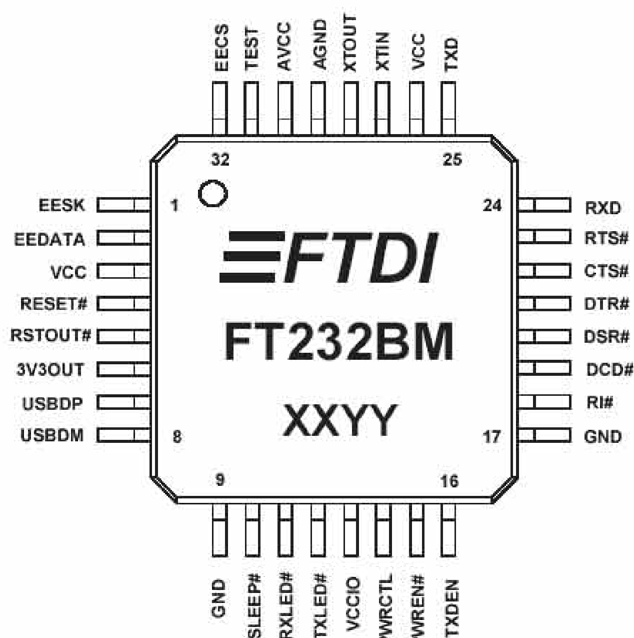
۷. با فشردن دکمه ی F5 برنامه را اجرا کنید. رشته ی Test با Baud Rate=1200، بدون بیت توازن، ۸ بیت داده و یک بیت پایان به COM1 ارسال خواهد شد.

پروژه ۱۳: ارتباط دهی USB با RS232

شماتیک:



Package چیپ FT232BM از نوع LQFP بوده و Pin out آن به صورت زیر می باشد:





پس از پیکربندی سخت افزاری مدار نیاز به درایور دارد که تنظیم آن در Win XP به صورت زیر است:

۱) درایور سخت افزار را می توانید از آدرس زیر دریافت نمایید:

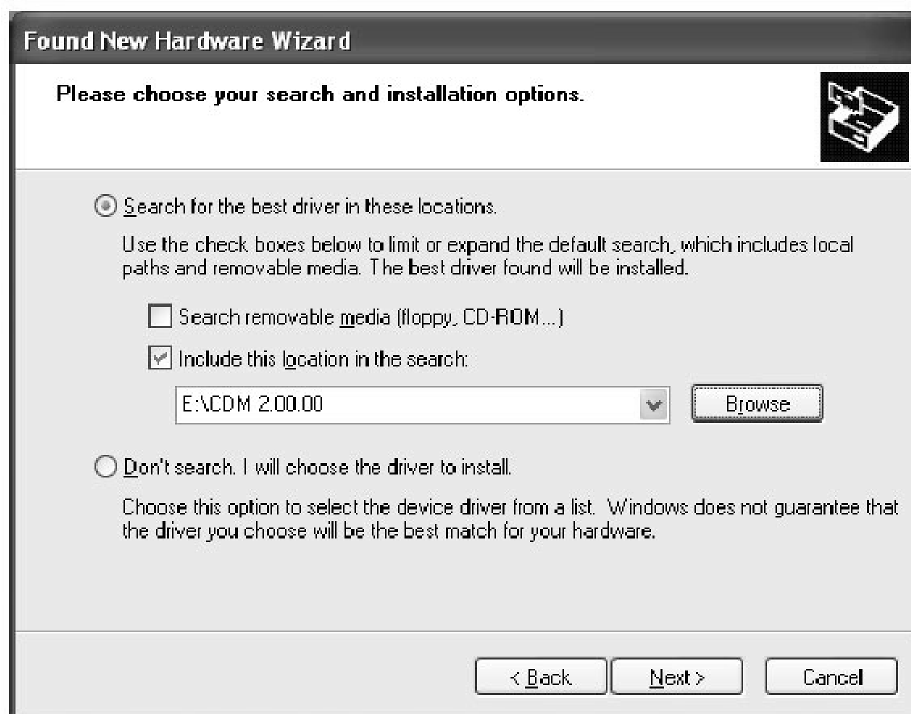
http://www.ftdichip.com/Drivers/VCP.htm

۲) فایل بارگذاری شده را در یک مسیر دلخواه Unzip کنید. (به عنوان مثال: E:\CDM 2.00.00)

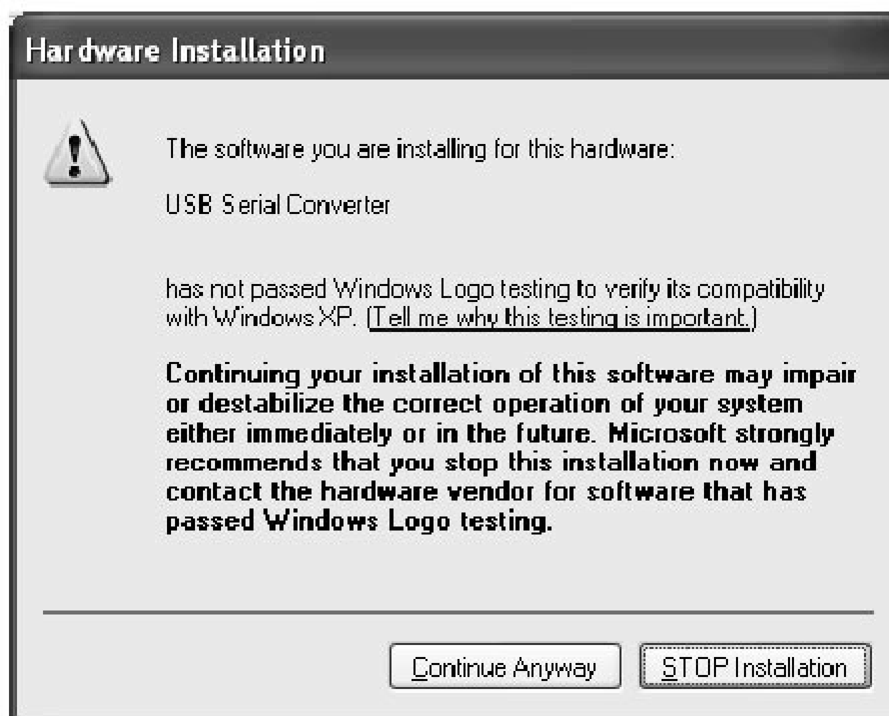
۳) سخت افزار خود را به یکی از کانکتورهای USB وصل کنید. در این حالت پیام **Found New Hardware** ظاهر شده و پس از آن کادر محاوره ی **Found New Hardware Wizard** نمایش داده می شود. گزینه ی دوم را انتخاب کرده و Next را کلیک کنید.



٤) در کادر حاصل مسیر درایور را وارد کرده و بر روی Next کلیک نمایید.



در صورتی که ویندوز به نحوی پیکربندی شده باشد که در صورت نصب درایورهای آزمایش نشده توسط مایکروسافت، پیغامی داده شود اخطار زیر ایجاد شده و در این حالت با چشم پوشی از اخطار بر روی Continue Anyway کلیک کنید.



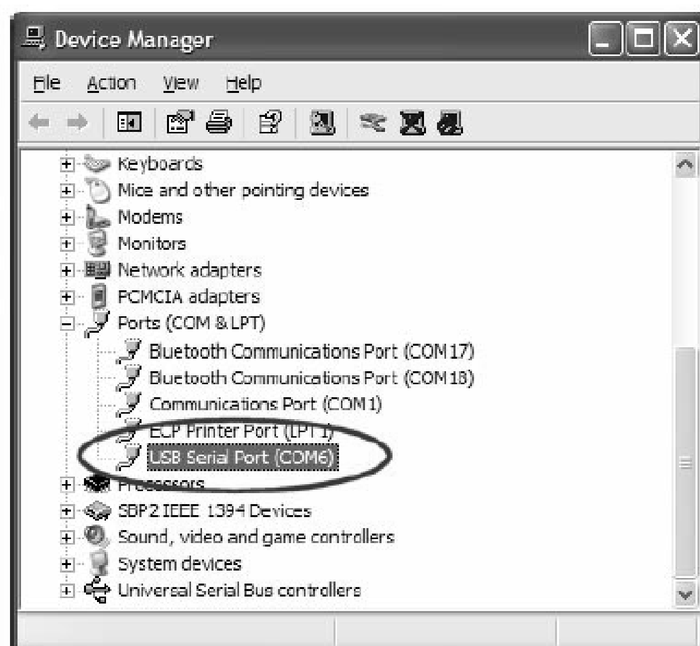
۵) کادر زیر نمایش داده شده و نشان می دهد که ویندوز در حال کپی نمودن فایل های مورد نیاز می باشد.



٦) ویندوز پیامی مبنی بر اتمام موفقیت آمیز نصب سخت افزار داده و با کلیک بر روی Finish این پروسه تمام می شود.



٧) Device Manager (در مسیر Control Panel\System بر روی زبانه ی Hardware) را باز کنید و درخت Port را با کلیک بر روی [+] کنار آن بسط دهید. ملاحظه می کنید که COM مجازی با نام COMx به لیست افزوده شده است. از این پس می توانید مطابق پروژه ی ١٢ اطلاعات را به COMx ارسال یا دریافت کرده و اطلاعات ارسال شده را در خروجی مدار دریافت کنید.


Device Manager
File Action View Help
Keyboards
Mice and other pointing devices
Modems
Monitors
Network adapters
PCMCIA adapters
Ports (COM & LPT)
Bluetooth Communications Port (COM17)
Bluetooth Communications Port (COM18)
Communications Port (COM1)
ECP Printer Port (LPT1)
USB Serial Port (COM6)
Processors
SBP2 IEEE 1394 Devices
Sound, video and game controllers
System devices
Universal Serial Bus controllers

(TWI) I^2C Bus

استاندارد I^2C در اوایل دهه ۱۹۸۰ توسط شرکت Philips طراحی شد. در ابتـدا ایـن پروتکـل بـه منظـور ایجـاد روشی ساده برای ایجاد ارتباط پردازنده با تراشه های جانبی در یک دستگاه تلویزیون ابداع شد.

I^2C طبق تعریف شرکت فیلیپس مخفف Inter-IC می باشد که بیانگر هدف آن یعنی فـراهم آوردن یـک لینـک ارتباطی بین مدارات مجتمع می باشد. امروزه این پروتکل به صـورت عمـومی در صـنعت پذیرفتـه شـده اسـت و کاربرد آن از سطح تجهیزات صوتی و تصویری نیز فراتر رفته است. به گونه ای که امروزه در بـیش از ۱۰۰۰ نـوع ICمختلف به کار گرفته شده است. I^2C فضا را حفظ می کند و باعث کاهش چشمگیر هزینه ی نهایی می شـود. دو خط ارتباطی به معنی Track های مسی کمتر و در نتیجه برد مدار چاپی کوچکتر و تست عیب یابی سـریعتر و راحتتر می باشد. علاوه بر این در اغلب موارد حساسیت مدار الکترونیکی نسبت به تداخل امـواج الکترومغناطیسـی و تخلیه ی الکتروستاتیکی کاهش می یابد.

برخی از ویژگی های باس I^2C

۱. I^2C یک پروتکل سریال سنکرون می باشد و کلاک آن می تواند بدون از دست رفتن اطلاعات تغییر کند.

۲. آدرس دهی ۷ بیتی (۱۲۷ وسیله ی متفاوت بر روی باس) و نیز آدرس دهـی ۱۰ بیتـی در ویـرایش جدیـد ایـن استاندارد (۱۰۲٤ وسیله بر روی باس).

۳. باس تنها به دو خط SCL و SDA نیاز دارد. بر روی SCL توسط Master کلاک ایجاد می شود و SDA خط Data ی دو طرفه می باشد که جهت آن توسط Master کنترل می شود.

٤. بر روی باس اطلاعات ۸ بیتی به صورت دو جهته با نرخ ارسال حداکثر ٤۰۰ کیلوبیت بر ثانیه. (و ۳.٤ مگابیت بر ثانیه در وضعیت High Speed)

۵. توانایی درایو نمودن تا ٤۰۰ پیکوفاراد ظرفیت خازنی تا فاصله ی حداکثر ٤ متر.

٦. حذف Spike های ناخواسته از خط SDA با استفاده از فیلتر موجود در چیپ، به طوری که سیگنال های ورودی که عرض آن ها کمتر از ۵۰ نانوثانیه باشد، حذف می شوند.

۷. پین های SDA و SCL مجهز به کنترل کننده ی Slew Rate می باشند که کارایی باس I^2C را در فرکانس های بالا (نزدیک ٤۰۰ کیلوهرتز) بهبود می بخشد. عملکرد کنترل کننده به این صورت است که لبه های تیز سیگنال را تا حدودی صاف کرده و در واقع با حذف هارمونیک های بالا به کاهش EMI کمک می کند.

۸. عدم نیاز به طراحی مدار واسط و راه اندازی باس تنها با دو مقاومت زیرا که مدار کنترل باس به صورت مجتمع بر روی وسیله ی I^2C قرار می گیرد.

اصطلاحاتی در این استاندارد

Master: وسیله ای است که شروع کننده ی ارسال اطلاعات و تولید کننده ی پالس کلاک و پایان دهنده ی ارسال اطلاعات می باشد.

Slave: وسیله ای است که توسط Master آدرس دهی شده است.

فرستنده: وسیله ای که اطلاعات را در باس داده قرار می دهد.

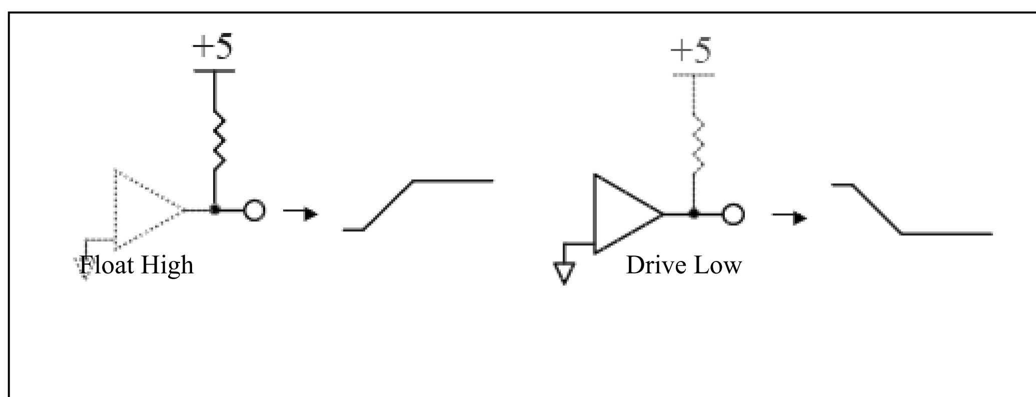
گیرنده: وسیله ای که اطلاعات را از باس می خواند.

Arbitration: مکانیسمی که اطمینان می دهد که اگر بیش از یک Master همزمان بخواهند باس را به کنترل خود درآورند، تنها یکی از آن ها بدون از دست رفتن اطلاعات موفق شود.

سطوح سیگنال در باس I^2C

به طور کلی دو سطح سیگنال در این استاندارد وجود دارد:

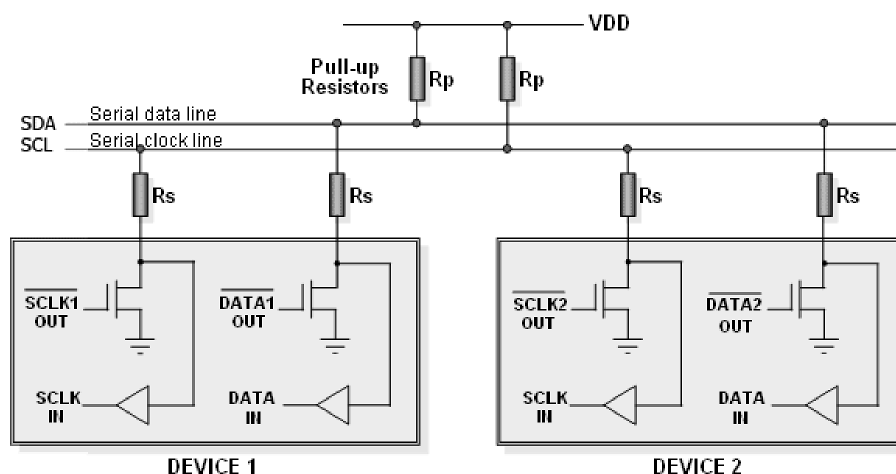
1. Float High (Logic 1)
2. Drive Low (Logic 0)



وضعیت idle یا بیکاری باس با سطح High Float مشخص می شود و با توجه با این مسئله نقش مقاومت های Pull-up مشخص می باشد که مقدار تقریبی آن ها با توجه به سرعت ارتباط به طور تقریبی از شکل زیـر بدسـت می آید:

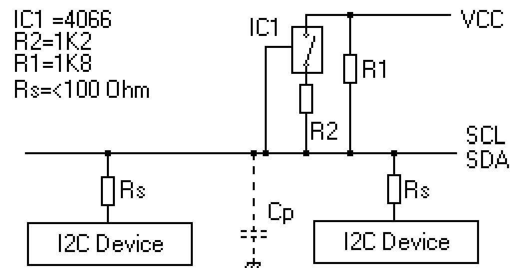
< 100 kbps	100 kbps	400 kbps
4.7 k	2.2 k	1 k

همانطور که در تصویر زیر مشاهده می کنید استفاده از یک مقاومـت Pull-up مشـترک باعـث ایجـاد یـک بـاس Wired-AND می شود، بدین معنا که برای صفر شدن باس تنها لازم است که یکی از وسایل خط را Low کند.



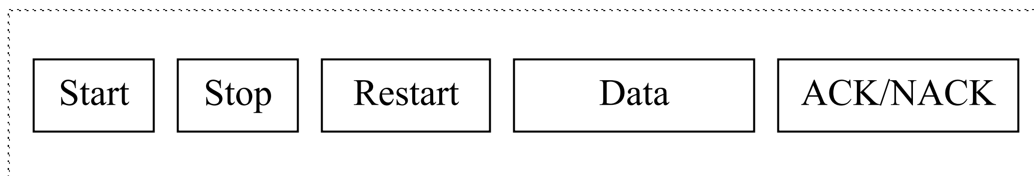
هر چند تکنیک ارائـه شـده در مـورد Open-collector بـودن و مقاومتهـای Pull-up دارای مزیـت Wired-AND می باشد ولی این موضوع در مورد خطوط طولانی که دارای یک ظرفیت خازنی مـی باشـند باعـث ایجـاد یک ثابت زمانی RC می گردد که برای رفع این موضوع به جای مقاومت می توان از Pull-up فعال و یا بافرهـای

ویژه ای که توسط شرکت فیلیپس ارائه شده است استفاده نمود. کاهش مقاومت در سطح High بوسیله ی یک سوییچ، باعث شارژ شریع خازن پارازیتی شده و در نتیجه زمان صعود و نزول پالس کاهش می یابد.

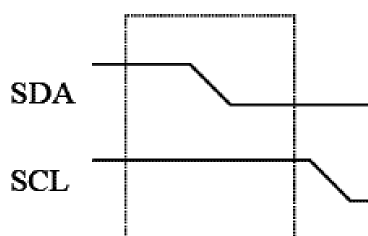


وضعیت های باس I^2C

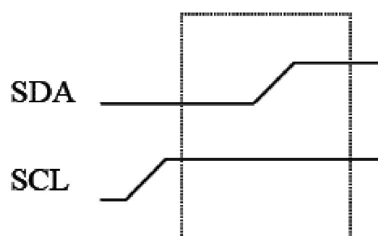
باس I^2C دارای تعدادی وضعیت یا عنصر می باشد که این وضعیت ها تعیین می کنند چه زمانی یک انتقال آغاز شود، خاتمه یابد، تاییدیه گرفته شود و غیره. شمای کلی این عناصر را در تصویر زیر مشاهده می کنید:



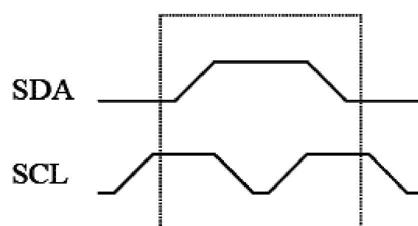
شرایط آغاز یا Start Condition: از زمانی که یک شرایط آغاز ایجاد می شود تا ایجاد یک شرایط پایان، باس Busy محسوب می شود. نمودار زمانی ایجاد این وضعیت به این صورت می باشد:



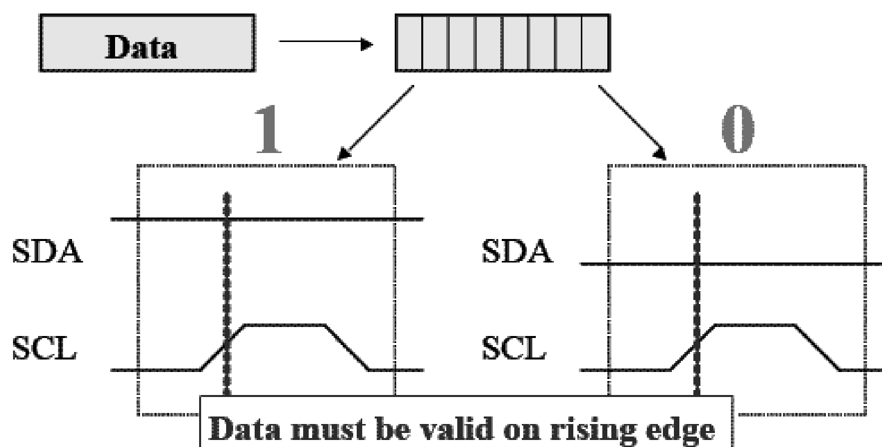
شرایط پایان یا Stop Condition: با ایجاد یک شرایط پایان باس آزاد شده و می تواند توسط یک Device دیگر حالت آغاز دیگری ایجاد شود:



شرایط شروع مجدد یا Restart Condition: این وضعیت زمانی کاربرد دارد که وسیله بخواهد بدون از دست دادن کنترل باس، شرایط آغاز دیگری ایجاد کند و ارتباط با Slave دیگری را آغاز کند. حالت شروع مجدد چیزی جز یک سگنال شروع که بعد از سیگنال پایان آمده است نمی باشد:



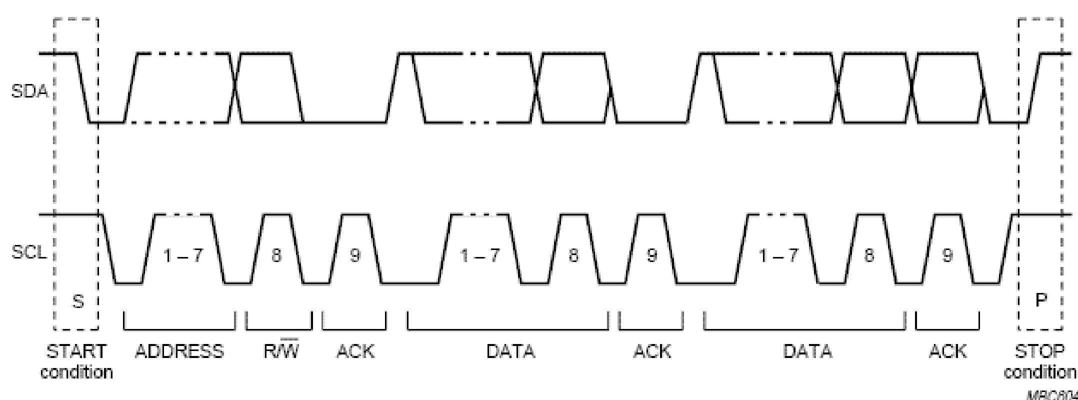
وضعیت Data: در این حالت ۸ بیت اطلاعات فرستاده می شود و هر بیت با یک کلاک همراهی می شود. خط SDA در لبه ی بالا رونده ی کلاک موجود در SCL نمونه برداری می شود و مقدار صفر یا یک منطقی از SDA خوانده می شود. به منظور اینکه مقدار خوانده شده از SDA معتبر باشد باید مقدار SDA در لبه ی بالا رونده ی SCL تغییر نکند و به طور کلی وضعیت خط SDA تنها در حالتی که SCL صفر است مجاز به تغییر است، تنها استثنا این قضیه شرایط آغاز و پایان است.



وضعیت ACK/NACK: وسیله ی گیرنده پس از دریافت ۸ بیت داده در سیکل نهم کلاک با زمین کردن خط SDA به فرستنده پاسخ یا Acknowledge می دهد. یک ماندن خط SDA در کلاک نهم نوعی عدم تصدیق Passive می باشد چرا که مقاومت Pull-up خط داده را یک نگاه می دارد.

قالب آدرس ۷ بیتی

پس از هر شرایط آغاز ۷ بیت آدرس وسیله ی مقصد فرستاده می شود که بعد از بیت هفتم یـک بیـت خوانـدن یـا نوشتن وجود دارد. صفر بودن این بیت تعیین می کند که Master قصد نوشتن روی وسیله را دارد و یک بودن آن نشان دهنده ی خواندن از وسیله ی مقصد می باشد.

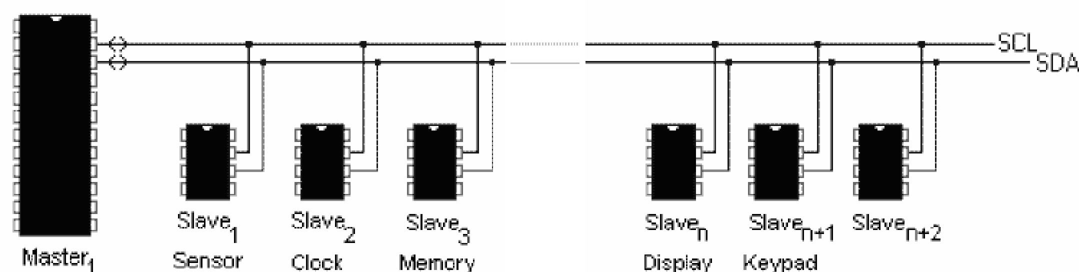


به طور کلی وسایل I^2C دارای یک آدرس ۷ بیتی می باشند که امکان آدرس دهی ۱۲۸ وسیله ی مختلف را فراهم می کند. از این تعداد ۱٦ آدرس رزرو شده می باشند و در نتیجه حداکثر ۱۱۲ وسیله بر روی یک باس قابل آدرس دهی هستند. آدرس صفر فراخوانی عمومی بوده و پس از ارسال این آدرس تمام میکروهایی که با زمین کردن SDA به Master پاسخ دهند، داده ی ۸ بیتی را دریافت خواهند کرد. پیغامی که برای همه Slave ها فرستاده می شود Broadcast نامیده می شود. این می تواند پیغامی باشد که توسط Master به تمامی Slave ها دستور داده می شود که آن وظیفه را انجام می دهند.

علاوه بر محدودیت ۱۱۲ آدرس، تعداد وسایل مجاز روی باس بوسیله ی ظرفیت خازنی ٤۰۰ پیکوفاراد نیز محدود می شود.

یک انتقال ساده

فرض کنید که در تصویر زیر میکروکنترلر Master می باشد و می خواهد اطلاعاتی را به یکی از Slave ها ارسال کند:



در ابتدا میکروکنترلر یک وضعیت شروع روی باس ایجاد می کند، این سیگنال همه ی Slave ها را متوجه باس می کند و پس از آن Master، آدرس Slave ای را که می خواهد با آن ارتباط برقرار کند را روی باس می فرستد و در بیت هشتم آن مشخص می سازد که می خواهد اطلاعات را از آن بخواند یا روی آن بنویسد. سپس تمام Slave ها آدرس را دریافت کرده و با آدرس خود مقایسه می کنند و وسیله ای که آدرس به آن تعلق داشته باشد با زمین کردن SDA در کلاک نهم یک سیگنال ACK ایجاد کرده و به Master پاسخ می دهد. به محض دریافت سیگنال تصدیق توسط Master تبادل اطلاعات با Slave آغاز شده و در نهایت Master، یک وضعیت Stop ایجاد می کند که به معنی آزاد شدن Bus می باشد.

درگاه TWI در میکروکنترلر AVR

واحد تولید نرخ بیت

این ماژول دوره ی تناوب خط SCL را در حالتی که به صورت Master عمل می کند، کنترل می نماید. این عمل بوسیله ی تنظیمات رجیستر TWBR و بیت های پیش تقسیم کننده در رجیستر TWSR انجام می شود. عملکرد Slave بستگی به تنظیمات نرخ بیت و پیش تقسیم کننده ندارد اما فرکانس CPU در Slave باید حداقل ۱۶ برابر فرکانس SCL باشد که از رابطه ی زیر بدست می آید:

$$f_{SCL} = \frac{\text{CPU Clock Frequency}}{16 + 2(TWBR) \times 4^{TWPS}}$$

توجه: در حالتی که TWI به صورت Master عمل می کند باید مقدار TWBR بزرگتر و یا مساوی ۱۰ باشد، در غیر اینصورت ممکن است Master کلاک اشتباه روی خط SCL تولید کند.

رجیسترهای TWI

بخش TWI دارای پنج رجیستر به نام های TWBR، TWCR، TWSR، TWDR و TWAR می باشد.

:TWI Bit Rate Register

Bit	7	6	5	4	3	2	1	0
TWBR	**TWBR[7:0]**							

این رجیستر فرکانس SCL را تعیین می کند و مقدار آن با توجه به رابطه ی زیر بدست می آید:

$$TWBR = \frac{\text{CPU Clock Frequency} - 16 \times f_{SCL}}{2 \times f_{SCK} \times ٤^{TWPS}}$$

مثال ۱: (تعیین TWBR با کریستال ۸ مگاهرتز و فرکانس SCK برابر ۱۰۰ کیلوهرتز و TWPS = 0)

$$\text{TWBR} = \frac{٨٠٠٠٠٠٠ - ١٦ \times ١٠٠٠٠٠}{٢ \times ١٠٠٠٠٠ \times ٤^{٠}} = ٣٢$$

:TWI Control Register

TWCR	7	6	5	4	3	2	1	0
نام بیت	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE

این رجیستر وظیفه ی کنترل TWI را بر عهده دارد و عملکرد بیت های آن به صورت زیر است:

TWI Interrupt Enable: با یک کردن این بیت در صورتی که بیت فعال ساز عمومی وقفه ها Set شده باشد، با فعال شدن پرچم TWINT برنامه به روتین سرویس وقفه ی TWI پرش خواهد کرد.

TWI Enable Bit: این بیت TWI را فعال کرده و با یک شدن آن، ماژول TWI کنترل پین های SCL و SDA را برعهده می گیرد. در این حالت واحد کنترل Slew Rate و فیلتر ورودی فعال شده و با صفر شدن این بیت TWI غیر فعال شده و مجددا به صورت I/O معمولی در می آید.

TWI Write Collision Flag: این بیت فقط خواندنی زمانی یک می شود که روی رجیستر TWDR مقداری نوشته شود و بیت TWINT صفر باشد. با نوشتن روی رجیستر TWDR در شرایطی که TWINT یک باشد، این پرچم پاک می شود.

TWI STOP Condition Bit: نوشتن یک روی این بیت در وضعیت Master باعث ایجاد یک شرایط پایان در باس I^2C خواهد شد و پس اجرا شدن این وضعیت بیت TWSTO به صورت خودکار پاک خواهد شد. در وضعیت Slave، یک کردن این بیت وضعیت پایان ایجاد نخواهد کرد اما خطوط SDA و SCL رها شده و به حالت شناور بر می گردند، بدین ترتیب می توان از وضعیت خطا خارج شد.

TWI START Condition Bit: نرم افزار با نوشتن یک روی این بیت با هدف Master شدن، ایجاد یک شرایط آغاز می کند. سخت افزار باس را چک کرده و در صورتی که آزاد باشد ایجاد یک شرایط آغاز می کند، در

غیر اینصورت ماژول TWI منتظر تشخیص یک شرایط پایان روی باس می شود و پس از آن با ایجاد یک شـرایط آغاز کنترل باس را به دست می گیرد. بیت TWSTA باید بعد از ایجاد شرایط آغاز توسط نرم افزار پاک شود.

TWI Enable Acknowledge Bit: TWEA ایجاد پالس تصدیق یا Acknowledge را کنترل می کنـد. اگر روی این بیت یک نوشته شود پالس ACK در شرایط زیر ایجاد می شود:

۱. پس از اینکه Slave آدرس مربوط به خودش را دریافت کرده باشد.

۲. مادامیکه بیت TWGCE در TWAR یک بوده و یک فراخوان عمومی دریافت شده باشد.

۳. یک بایت داده در Master یا Slave دریافت شده باشد.

TWI Interrupt Flag: این بیت پس از اتمام هر یک از وظایف TWI یک شده و در صـورتی کـه وقفـه ی سراسری و بیت TWIE یک باشند به بردار وقفه ی TWI پرش مـی کنـد. ایـن بیـت پـس از اجـرای ISR بـه صورت سخت افزاری پاک نمی شود و باید توسط نرم افزار با نوشتن یک پاک شود.

هنگامی که پرچم TWINT یک شود، نرم افزار باید محتویات تمام رجیسترهایی را که برای وظیفه ی بعدی نیـاز است (مثل TWDR) تنظیم کرده و پس از آن همزمان با تنظیمات رجیسـتر TWCR، پـرچم TWINT را پـاک کند تا ماژول TWI شروع به انجام وظیفه ی بعدی نمایـد. مادامیکـه کـه پـرچم TWINT یـک اسـت SCL در وضعیت صفر باقی می ماند و تا زمانی که این بیت پاک نشود TWI هیچ عملی انجام نخواهد داد.

TWI Status Register:

TWSR	7	6	5	4	3	2	1	0
نام بیت	TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0

TWI Prescaler Bits[1:0]: این بیت ها مقدار پیش تقسیم کننده را مشخص می کنند که مطابق جدول زیر تعیین می شود:

TWPS1	TWPS0	مقدار پیش تقسیم کننده
۰	۰	۱
۰	۱	٤
۱	۰	١٦
۱	۱	٦٤

TWI Status[7:3]: این بیت ها نمایانگر وضعیت TWI می باشند که در بررسی Mode های کاری عملکرد آن ها بررسی خواهد شد.

TWI Data Register:

Bit	7	6	5	4	3	2	1	0
TWDR	**TWDR[7:0]**							

در حالت انتقال، TWDR محتوی بایت بعدی است که باید ارسال شود و در حالت دریافت شامل آخرین بایت دریافتی است. این رجیستر تنها در حالتی که TWI در حال انتقال اطلاعات نباشد و پرچم TWINT یک شده باشد، قابل نوشتن است.

TWI (Slave) Address Register: این رجیستر آدرس Slave در وضعیت های ST و SR می باشد.

TWAR	7	6	5	4	3	2	1	0
نام بیت	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE

TWI General Call Recognition Enable: آدرس 0x00 مربوط به فراخوانی عمومی بوده و تشخیص آن توسط سخت افزار منوط به یک بودن TWGCE می باشد.

TWI (Slave) Address[7:1]: این ۷ بیت آدرس Slave را مشخص می کنند.

نام گذاری های زیر به صورت قرارداد در ادامه استفاده خواهند شد:	
S:	START condition
Rs:	REPEATED START condition
R:	Read bit (high level at SDA)
W:	Write bit (low level at SDA)
ACK:	Acknowledge bit (low level at SDA)
NACK:	Not acknowledge bit (high level at SDA)

Data: 8-bit data byte

P: STOP condition

SLA: Slave Address

Mode های عملکرد TWI:

1. Master Transmitter (MT)
2. Master Receiver (MR)
3. Slave Transmitter (ST)
4. Slave Receiver (SR)

(MT) Master Transmitter ①

در این Mode ابتدا توسط Master یک وضعیت شروع ایجاد شده و بلوک آدرسی که در ادامه فرستاده می شود تعیین می کند که Master گیرنده یا فرستنده است. اگر پس از ایجاد حالت شروع SLA+W ارسال گردد Master وارد حالت ارسال شده و در صورتی که SLA+R ارسال شود وارد حالت دریافت می شود.

شرایط آغاز با مقدار دهی TWCR به صورت زیر ایجاد می گردد:

TWCR	7	6	5	4	3	2	1	0
نام بیت	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
مقدار	1	X	1	0	X	1	0	X

TWIE فعال ساز وقفه ی TWI بوده و مقدار آن اختیاری می باشد.

TWEN فعال ساز TWI بوده و در نتیجه باید مقدار یک داشته باشد.

TWWC که یک بیت فقط خواندنی است در این حالت صفر خوانده می شود زیرا نرم افزار هنوز مقـداری را روی TWDR ننوشته است و بنابراین Collision یا تصادمی نیز پیش نیامده است.

TWSTO صفر می باشد زیرا در اینجا نمی خواهیم شرایط پایان ایجاد شود.

TWSA یک بوده تا ماژول TWI در صورت آزاد بودن باس، شرایط آغـاز را ایجـاد کـرده و در غیـر اینصـورت منتظر تشخیص یک شرایط پایان باشد.

TWEA مقداری بی اهمیت می باشد زیرا با فرض Set کردن آن در شرایط آغاز هیچ گـاه ACK نیـاز نبـوده و ایجاد نیز نمی شود.

TWINT با نوشتن یک پاک می شود تا TWI شروع به انجام فرمان داده شده توسط رجیستر TWCR کند.

پس از اجرای تنظیمات فوق در صـورت آزاد بـودن بـاس مـاژول TWI شـرایط آغـاز را ایجـاد کـرده و در غیـر اینصورت منتظر آزاد شدن باس شده و به محض تشخیص یک شرایط پایان کنترل باس را به دست گرفته و شرایط آغاز را ایجاد می نماید، پس از آن پرچم TWINT یک شـده و کـد وضـعیت موجـود در TWSR برابـر 0x08 خواهد شد. برای ورود به وضعیت MT باید روی TWDR مقدار SLA+W نوشته شده و پس از آن با نوشـتن یک روی TWINT آن را پاک کرده تـا مرحلـه ی بعـد آغـاز شـود. پـس از نوشـتن SLA+W روی TWDR تنظیمات TWCR به صورت زیر خواهد بود:

TWCR	7	6	5	4	3	2	1	0
نام بیت	TWIN	TWE	TWST	TWST	TWW	TWE	-	TWI

	T	A	A	O	C	N		E
مقدار	1	X	0	0	X	1	0	X

تفاوت با مرحله ی قبل این است که از آنجایی که نیاز به شرایط آغاز وجود ندارد بیت TWSTA صفر می باشد. با فرستاده شدن SLA+W و دریافت ACK مجددا بیت TWINT یک شده و در این شرایط یکی از چند کد 0x20، 0x18 یا 0x38 در TWSR قرار می گیرد. در صورتی که این مقدار برابر 0x18 باشد بدین معناست که آدرس ارسال شده و ACK دریافت شده است. پس از این باید بلوک داده ارسال شود. همانند حالت قبل ابتدا مقدار مورد نظر در رجیستر TWDR نوشته شده و با پاک کردن بیت TWINT داده ی ۸ بیتی توسط TWI ارسال خواهد شد. در صورتی که رجیستر TWDR بعد از پاک کردن TWINT مقداردهی شود رجیستر TWWC یک شده و اعلام یک حالت تصادم را می کند. تنظیمات TWCR همانند حالت قبل به صورت زیر خواهد بود:

TWCR	7	6	5	4	3	2	1	0
نام بیت	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
مقدار	1	X	0	0	X	1	0	X

این مرحله تا اتمام ارسال تمام بایت های داده تکرار می شود و سرانجام با ایجاد یک شرایط پایان یا شروع مجدد پایان می یابد. شرایط پایان با مقداردهی رجیستر TWCR به صورت زیر ایجاد می شود:

TWC	7	6	5	4	3	2	1	0

R								
نام بیت	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
مقدار	1	X	0	1	X	1	0	X

و برای ایجاد شرایط شروع مجدد جدول زیر را خواهیم داشت:

TWCR	7	6	5	4	3	2	1	0
نام بیت	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
مقدار	1	X	1	0	X	1	0	X

✓ پس از ایجاد شرایط پایان TWINT0 یک نمی شود.

با ایجاد یک حالت شروع مجدد مقدار رجیستر TWSR برابر 0x10 خواهد بود و این قابلیت به Master اجازه تغییر Slave را بدون از دست دادن کنترل باس می دهد.

مقادیر رجیستر TWSR در وضعیت های مختلف حالت Master Transmitter مطابق جدول زیر می باشد:

مقدار TWSR	عملکرد	وضعیت بعدی که می تواند بوسیله ی TWI انجام شود
0x08	وضعیت شروع ایجاد شده است.	ارسال SLA+W و دریافت ACK یا NACK
0x10	وضعیت شروع دوباره ایجاد شده است.	ارسال SLA+W و دریافت ACK یا NACK ارسال SLA+R و دریافت ACK یا NACK

0x18	بایت SLA+W ارسال شده و ACK دریافت شده است.	ارسال بایت داده و دریافت ACK یا NACK ایجاد حالت شروع مجدد ایجاد حالت پایان
0x20	بایت SLA+W ارسال شده و ACK دریافت نشده است.	ایجاد حالت شروع مجدد ایجاد حالت پایان
0x28	بایت داده ارسال شده و ACK دریافت شده است.	ارسال بایت داده و دریافت ACK یا NACK ایجاد حالت شروع مجدد ایجاد حالت پایان
0x30	بایت داده ارسال شده و ACK دریافت نشده است.	ارسال بایت داده و دریافت ACK یا NACK ایجاد حالت شروع مجدد ایجاد حالت پایان
0x38	کنترل باس از دست رفته است.	ارسال حالت شروع و تلاش برای کنترل مجدد آن

مثال۲: (ارسال عدد 0x77 به Slave با آدرس 0xA0 و با نرخ بیت ۱۰۰ کیلوهرتز)

```
#include <mega16.h>
#define xtal 8000000

void main()
{

//--- Start Condition and Transmitting SLA+W ----

  TWBR = 32; // Bit rate = 100Khz
  TWCR = 0xA4; // Transmit Start Condition
```

```
  while(TWCR&0x80==0); // Waiting for TWINT flag

  if(TWSR&0xF8==0x08) // Start Condition Transmitted?
  {
        TWDR=0xA0; // SLA+W
        TWCR=0x84; // Enable TWI and Clear TWINT
  }
  else
        goto error;

//--------------- Transmitting Data -------------

  while(TWCR&0x80==0); // Waiting for TWINT flag

  if(TWSR&0xF8==0x18) // SLA+W has been send with ACK?
  {
        TWDR=0x77; // Data=0x77
        TWCR=0x84; // Enable TWI and Clear TWINT
  }
  else
        goto error;

//--------- Transmitting Stop Condition ----------

  while(TWCR&0x80==0); // Waiting for TWINT flag

  if(TWSR&0xF8==0x28) // Data has been send with ACK?
        TWCR=0x94; // Transmit Stop Condition
```

```
//-----------------------------------------------

error:
  while(1);

}
```

(MR) Master Receiver ②

در این وضعیت Master اطلاعات را از Slave دریافت می کند و برای ایجاد آن باید پس از ایجاد شـرایط آغـاز بر خلاف حالت قبل SLA+R به Slave فرستاده شود. در صورتی که شرایط آغاز به درسـتی ایجـاد شـده باشـد بیت TWINT یک شده و مقدار TWSR برابر 0x08 می شـود، بـرای ارسـال SLA+R بایـد مقـدار آن را در TWDR بارگذاری کرده و با پاک کردن TWINT آن را ارسال نمود.

زمانی که SLA+R فرستاده شود و ACK دریافت شود بیـت TWINT یـک شـده و در ایـن شـرایط یکـی از کدهای 0x38، 0x40 یا 0x48 ممکن است در TWSR قرار گیرد:

مقدار TWSR	عملکرد	وضعیت بعدی که می تواند بوسیله ی TWI انجام شود
0x08	وضعیت شروع ایجاد شده است.	ارسال SLA+R و دریافت ACK یا NACK
0x10	وضعیت شروع دوباره ایجاد شده است.	ارسال SLA+R و دریافت ACK یا NACK ارسال SLA+W و دریافت ACK یا NACK
0x38	کنترل باس از دست رفته است.	ارسال حالت شروع و تلاش برای کنترل مجدد آن
0x40	بایت SLA+R ارسال شده و ACK دریافت شده است.	دریافت بایت داده و ارسال ACK یا NACK
0x48	بایت SLA+R ارسال شده و ACK دریافت نشده است.	ایحاد وضعیت شروع مجدد ایجاد وضعیت پایان

0x50	دریافت بایت داده و ارسال ACK	دریافت بایت داده و ارسال ACK یا NACK
0x58	دریافت بایت داده و ارسال NACK	ایحاد وضعیت شروع مجدد ایجاد وضعیت پایان

در صورتی که SLA+R ارسال شده و ACK دریافت شده باشد بعد از یـک شـدن TWINT داده ی دریافـت شده در TWDR می تواند خوانده شود. این کار تا دریافت آخرین بایـت تکـرار شـده و بعـد از آن Master بـا فرستادن NACK به Slave اعلام می کند که دیگر قصد خواندن از آن را نـدارد و در نتیجـه مـی تـوان عملیـات خواندن را با ایجاد یک شرایط شروع مجدد یا پایان متوقف نمود.

مثال ۳: (خواندن ۲ بایت داده از Slave با آدرس 0xA0 و نرخ بیت ۱۰۰ کیلوهرتز)

```
#include <mega16.h>
#define xtal 8000000

char incoming_data;

void main()
{

//----------- Sending Start Condition -----------

  TWBR = 32; // Bit rate = 100Khz
  TWCR = 0xA4; // Transmit Start Condition

  while(TWCR&0x80==0); // Waiting for TWINT flag

//-------------- Sending SLA + R ----------------
```

```
  if(TWSR&0xF8==0x08) // Start Condition Transmitted?
  {
        TWDR=0xA1; // SLA+R
        TWCR=0xC4; // Enable TWI and Clear TWINT
  }
  else
        goto error;

//-------- Enable Master Acknowledging ----------

  while(TWCR&0x80==0); // Waiting for TWINT flag

  if(TWSR&0xF8==0x40) // SLA+R has been send with ACK?
     TWCR=0xC4; // Master Acknowledging and clear TWINT

//--------------- Reading 1st byte --------------

  while(TWCR&0x80==0); // Waiting for TWINT flag

  if(TWSR&0xF8==0x50) // Master has been Received Data?
  {
     incoming_data = TWDR; // Reading Data
     TWCR=0xC4; // Master Acknowledging and clear TWINT
  }
  else
        goto error;

//---- Reading 2nd byte and Stop Condition ------
```

```
  while(TWCR&0x80==0); // Waiting for TWINT flag

  if(TWSR&0xF8==0x50) // Master has been Received Data?
  {
     Incoming_data=TWDR;
     TWCR=0x94; // Master Not Acknowledging and clear
TWINT
  }
  else
        goto error;

        TWCR=0x94; // Transmit Stop Condition

//-------------------------------------------------

error:
  while(1);

}
```

③ Slave Receiver (SR)

در این وضعیت میکروکنترلر گیرنده، Slave بوده و اطلاعـات را از Master Transmitter دریافـت مـی کنـد. برای شروع به کار باید آدرس Slave در ۷ بیت بـالای TWAR قـرار گیـرد. و چنانچـه بیـت LSB یـک شـود Master قادر به پاسخگویی فراخوانی عمومی نیز خواهد بود و در غیر اینصورت آن را نادیده خواهد گرفت.

TWAR	7	6	5	4	3	2	1	0
نام بیت	TWAR[7:1]							X

TWCR مطابق زیر مقدار دهی شود:

TWCR	7	6	5	4	3	2	1	0
نام بیت	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
مقدار	1	1	0	0	0	1	0	X

بیت TWIE یک شده تا ماژول TWI فعال شود.

بیت TWEA یک شده تا ارسال ACK فعال شود.

بیت های TWSTA و TWSTO صفر می باشند زیرا ایجاد شرایط آغاز و پایان بر عهده ی Master می باشد.

بعد از پیکربندی Slave، منتظر شده تا توسط Master آدرس دهی شده و یا یک فراخوان عمومی دریافت کند. اگر در آدرس دریافت شده بیت جهت (W/R) صفر باشد نشان دهنده ی این است که Master می خواهد مقداری را به Slave بفرستد که در اینصورت Slave وارد وضعیت Slave Read (SR) شده و در غیر اینصورت در حالت Slave Transmiter (ST) قرار خواهد گرفت. پس از دریافت SLA+W بیت TWINT یک شده و وضعیت جاری در رجیستر TWSR وجود خواهد داشت:

مقدار TWSR	عملکرد	وضعیت بعدی که می تواند بوسیله ی TWI انجام شود
0x60	بایت SLA+W دریافت شده و ACK ارسال شده است.	دریافت بایت داده و ارسال ACK یا NACK
0x68	کنترل باس در حین ارسال SLA + W توسط Master از دست رفته است.	دریافت بایت داده و ارسال ACK یا NACK

0x70	فراخوانی عمومی ریافت شده و ACK ارسال شده است.	دریافت بایت داده و ارسال ACK یا NACK
0x78	کنترل باس در حین ارسال فراخوانی عمومی توسط Master از دست رفته است.	دریافت بایت داده و ارسال ACK یا NACK
0x80	دریافت بایت داده و ارسال ACK در فراخوانی اختصاصی	دریافت بایت داده و ارسال ACK یا NACK
0x88	دریافت بایت داده و ارسال NACK در فراخوانی اختصاصی	تغییر به حالت بدون آدرس و یا ارسال حالت شروع برای تغییر وضعیت از Slave به Master
0x90	دریافت بایت داده و ارسال ACK در فراخوانی عمومی	دریافت بایت داده و ارسال ACK یا NACK
0x98	دریافت بایت داده و ارسال NACK در فراخوانی عمومی	تغییر به حالت بدون آدرس و یا ارسال حالت شروع برای تغییر وضعیت از Slave به Master
0xA0	دریافت حالت پایان و یا شروع دوباره	تغییر به حالت بدون آدرس و یا ارسال حالت شروع برای تغییر وضعیت از Slave به Master

در صورتی که کد خوانده شده 0x60 باشد آدرس بدرستی توسط Slave دریافت شده و برای Master تصدیق (ACK) نیز ارسال شده است. پس از این Slave می تواند بایت های داده را دریافت کرده و تا زمانی که یک شرایط پایان یا شروع مجدد ایجاد شود این روند ادامه خواهد داشت که در این صورت مقدار رجیستر TWSR برابر 0xA0 می باشد.

مثال ٤: (دریافت یک بایت داده توسط Slave با آدرس 0x01 و نرخ بیت ۱۰۰ کیلوهرتز)

```
#include <mega16.h>
#define xtal 8000000

char incoming_data;

void main()
{
```

```
//-------------- Initial Setting ---------------

  TWAR = 0x01; // Slave Address
  TWBR = 32; // Bit rate = 100Khz
  TWCR = 0xC4; // Clear Int, Set TWEA and TWEN

  while(TWCR&0x80==0); // Waiting for TWINT flag

//--------- Transmit a Byte to Master ----------

  if(TWSR&0xF8==0x60) // SLA+W Received?
  {
        TWCR = 0xC4; // Clear Int, Set TWEA and TWEN
  }
  else
        goto error;

//------------ Slave Receive a Byte ------------

  while(TWCR&0x80==0); // Waiting for TWINT flag

  if(TWSR&0xF8==0x80) // Slave has been Receive Byte?
  {
       incoming_data=TWDR; //Receive incoming data
       TWCR=0x84; // Clear TWINT
  }
  else
     goto error;
```

```
//---------- Receiving Stop Condition ----------

  while(TWCR&0x80==0); // Waiting for TWINT flag

  if ((TWSR & 0xF8)== 0x0A0) //Stop Condition Received?
       TWCR=0x84; // Clear TWINT

error:
  while(1);

}
```

④ Slave Transmitter (ST)

این وضعیت مشابه SR می باشد با این تفاوت که بعد از دریافت آدرس بیت W/R یک می باشد. بدین ترتیب Master اعلام می کند که می خواهد مقداری را از Slave بخواند و در نتیجه Slave فرستنده خواهد بود.

پس از دریافت آدرس بیت TWINT یک شده و می توان کد وضعیت جاری را از TWSR بدست آورد، بعد از ارسال هر بایت داده از Slave، Master با فرستادن ACK آن را تصدیق می کند و چنانچه Master بخواهد آخرین بایت را در دریافت کند NACK ارسال خواهد نمود (TWSR=0xC0).

مقدار TWSR	عملکرد	وضعیت بعدی که می تواند بوسیله ی TWI انجام شود
0xA8	بایت SLA+R دریافت شده و ACK ارسال شده است.	ارسال بایت داده و دریافت ACK یا NACK

0xB0	کنترل باس در حین ارسال SLA + W توسط Master از دست رفته است.	ارسال بایت داده و دریافت ACK یا NACK
0xB8	ارسال بایت داده و دریافت ACK	ارسال بایت داده و دریافت ACK یا NACK
0xC0	ارسال بایت داده و دریافت NACK	تغییر به حالت بدون آدرس و یا ارسال حالت شروع برای تغییر وضعیت از Slave به Master
0xC8	ارسال آخرین بایت داده و دریافت ACK (TWEA=0)	تغییر به حالت بدون آدرس و یا ارسال حالت شروع برای تغییر وضعیت از Slave به Master

مثال ۵: (خواندن یک بایت داده از Slave با آدرس 0x01 و نرخ بیت ۱۰۰ کیلوهرتز)

```
#include <mega16.h>
#define xtal 8000000

void main()
{

//-------------- Initial Setting ---------------

  TWAR = 0x01 // Slave Address
  TWBR = 32; // Bit rate = 100Khz
  TWCR = 0xC4; // Clear Int, Set TWEA and TWEN

  while(TWCR&0x80==0); // Waiting for TWINT flag

//--------- Transmit a Byte to Master ----------

  if(TWSR&0xF8==0xA8) // SLA+R Received?
  {
        TWDR=0x77; // Transmitt 0x77
```

```
        TWCR = 0xC4; // Clear Int, Set TWEA and TWEN
  }
  else
        goto error;

//------------- Wait for ACK or NACK -----------

  while(TWCR&0x80==0); // Waiting for TWINT flag

  if(TWSR&0xF8!=0xC0) // Slave has been send NACK?
     goto error;

//-----------------Error Sub -------------------

error:
  while(1);

}
```

همزمان سازی پالس ساعت:

پالس ساعت توسط Master ها تولید می گردد. هر Master پالس ساعت خود را بر روی SCL قرار می دهد و با توجه به خاصیت wired-AND در باس I^2C پالس ساعت ها با هم AND شده و باعث تولید یک پالس ساعت مشترک می گردد.

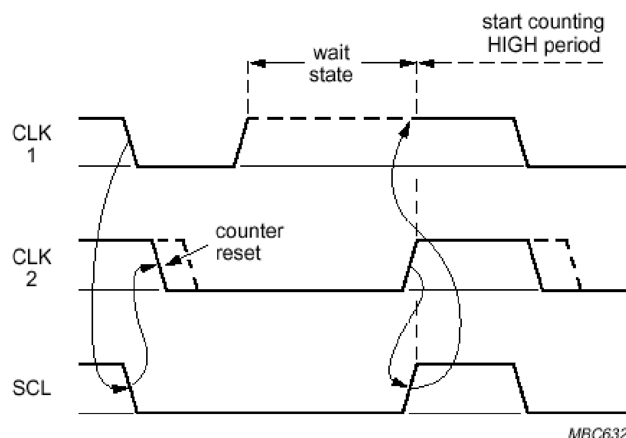


Fig.8 Clock synchronization during the arbitration procedure.

Arbitration در سیستم های دارای چند Master

هر Master تنها در زمانی می تواند به باس دسترسی پیدا کند کـه خـط SDA آزاد باشـد. امـا پروتکـل I^2C بـه شکلی طراحی شده است که در صورتیکه در شرایط آزاد بـودن بـاس دو یـا چنـد Master همزمـان درخواسـت دسترسی به باس را داشته باشند بدون از دست رفتن اطلاعات ارتباط حفـظ شـود. در اینجـا نیـز وجـود خاصـیت Wired-AND باعث حل مشکل می گردد یعنی چند Master بطور همزمان داده هایشان را بر روی خط SDA به صورت سریال ارسال می دارند که این بیت ها با هم AND شده و بر روی باس یک دیتای واحد را ایجاد مـی کند، در اولین مکانی که خط SDA با خط داده مربوط به یک Master مطابقـت نداشـت آن Master خـط داده سریال را در سطح یک منطقی رها می کند (حالت پیش فرض با توجه به وجود Pull-up سطح یک می باشد) تـا بر روی کار دیگر Master ها اخلالی ایجاد نکند.

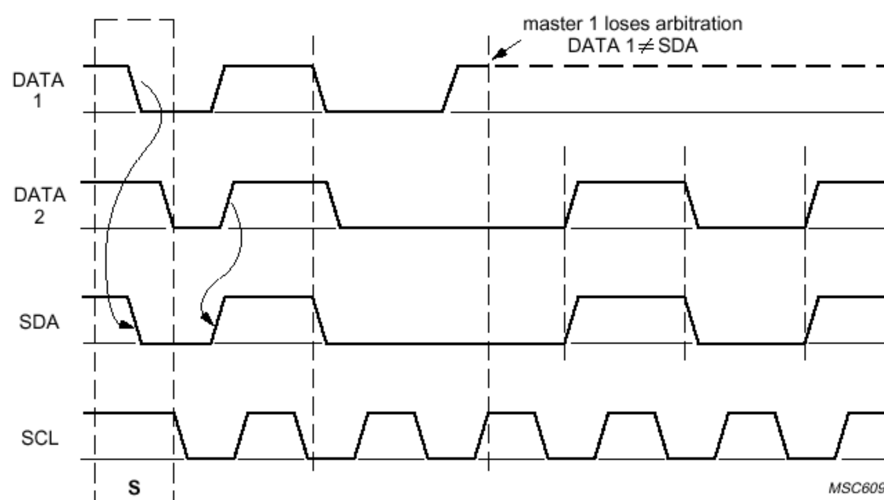


Fig.9 Arbitration procedure of two masters.

همان طور که دیده می شود مساله Arbitration تنها در مورد حالتی معنی دارد که چنـد Master داشـته باشـیم زیرا:

۱. در مورد Slave ها با توجه به اینکه در هر زمان یـک Slave آدرس دهـی مـی شـود و حـق دسترسـی بـه SDA را دارد معنی نخواهد داشت.

۲. یک Master دیگر رقیبی برای دسترسی به خط SDA ندارد.

دسترسی نرم افزاری به I^2C در CodeVision

کامپایلر CodeVision با ارائه ی یک سری توابع مربوط به باس I^2C، امکان ایجاد این پروتکل را به صورت نرم افزاری به برنامه نویس می دهد. پین های SDA و SCL باید توسط نرم افزار به صورت زیر تعیین شوند:

```
#asm
```

```
    .equ __i2c_port=0x18
    .equ __sda_bit=3
    .equ __scl_bit=4

#endasm
```

در این قطعه کد پین های ۳ و ٤ از PORTB به عنوان SDA و SCL تعیین شده اند.

توابع I^2C در کامپایلر CodeVision

:i2c_init()

این تابع تنظیمات اولیه ی باس I^2C را انجام داده و باید قبل از استفاده از توابع دیگر به به کار برده شود.

:i2c_start()

این تابع یک شرایط آغاز ایجاد می کند و در صورتی که باس آزاد باشد مقدار یک را برمی گرداند و در غر این صورت خروجی این تابع صفر خواهد بود.

:i2c_stop()

این تابع یک شرایط پایان بر روی باس I^2C ایجاد می کند.

:i2c_read()

این تابع یک بایت را از باس I2C خوانده و شکل کلی آن به صورت زیر می باشد:

```
unsigned char i2c_read(unsigned char ack)
```

پارامتر **ack** تعیین می کند که آیا پس از دریافت یک بایت **acknowledgement** ارسال شود یا خیر. در صورتی که این پارامتر یک باشد ACK ارسال خواهد و در غیر اینصورت با ایجاد نکردن ACK به صورت پسیو NACK ایجاد خواهد شد.

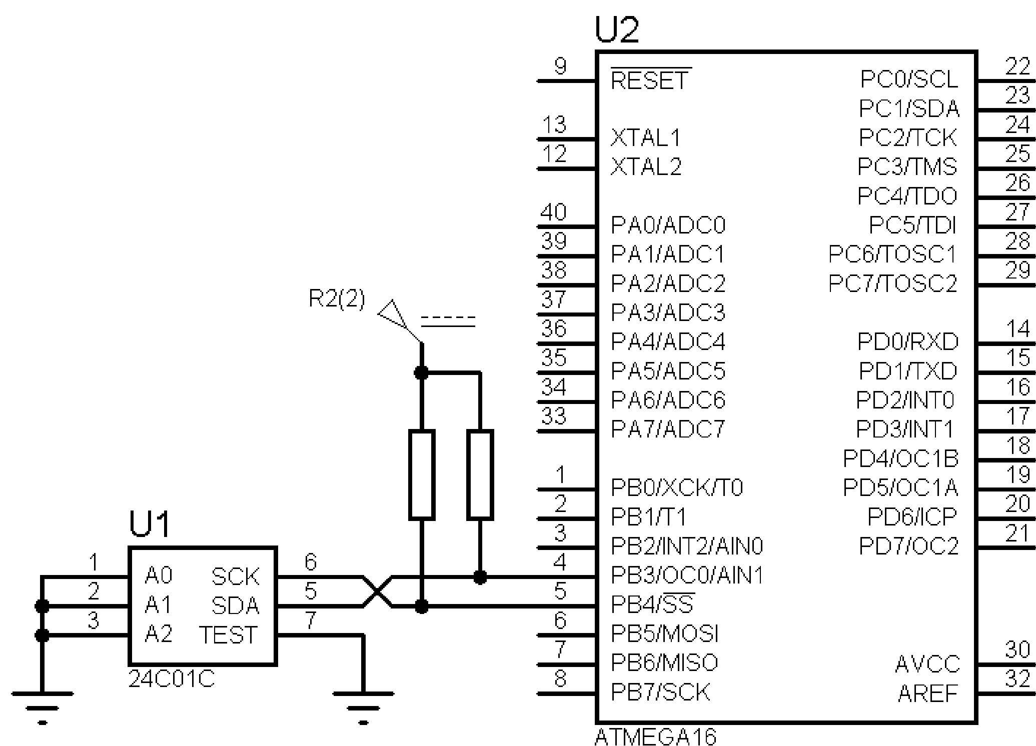
:i2c_write()

این تابع یک بایت را به باس I^2C ارسال کرده و شکل کلی آن به صورت زیر می باشد:

```
unsigned char i2c_write(unsigned char data)
```

متغیر **data** مقدار ارسالی به باس بوده و در صورتی که Slave، ACK ایجاد کند این تابع مقدار یک و در غیر اینصورت مقدار صفر باز می گرداند.

شماتیک:



نرم افزار:

```
#include<mega16.h>
#define xtal 1000000

/* the I2C bus is connected to PORTB */
/* the SDA signal is bit 3 */
/* the SCL signal is bit 4 */

#asm

    .equ __i2c_port=0x18
    .equ __sda_bit=3
```

```
    .equ __scl_bit=4

#endasm

/* now you can include the I2C Functions */

#include <i2c.h>

/* function declaration for delay_ms */
#include <delay.h>

#define EEPROM_BUS_ADDRESS 0xa0

/* read a byte from the EEPROM */
unsigned char eeprom_read(unsigned char address) {
unsigned char data;
i2c_start();
i2c_write(EEPROM_BUS_ADDRESS);
i2c_write(address);
i2c_start();
i2c_write(EEPROM_BUS_ADDRESS | 1);
data=i2c_read(0);
i2c_stop();
return data;
}

/* write a byte to the EEPROM */
void eeprom_write(unsigned char address, unsigned char
data) {
```

```
i2c_start();
i2c_write(EEPROM_BUS_ADDRESS);
i2c_write(address);
i2c_write(data);
i2c_stop();

/* 10ms delay to complete the write operation */
delay_ms(10);
}

void main(void) {
unsigned char i;
DDRD=0xFF;

/* initialize the I2C bus */
i2c_init();

/* write the byte 55h at address 10h */
eeprom_write(0x10,0x55);

/* read the byte from address AAh */
i=eeprom_read(0x10);
PORTD=i;

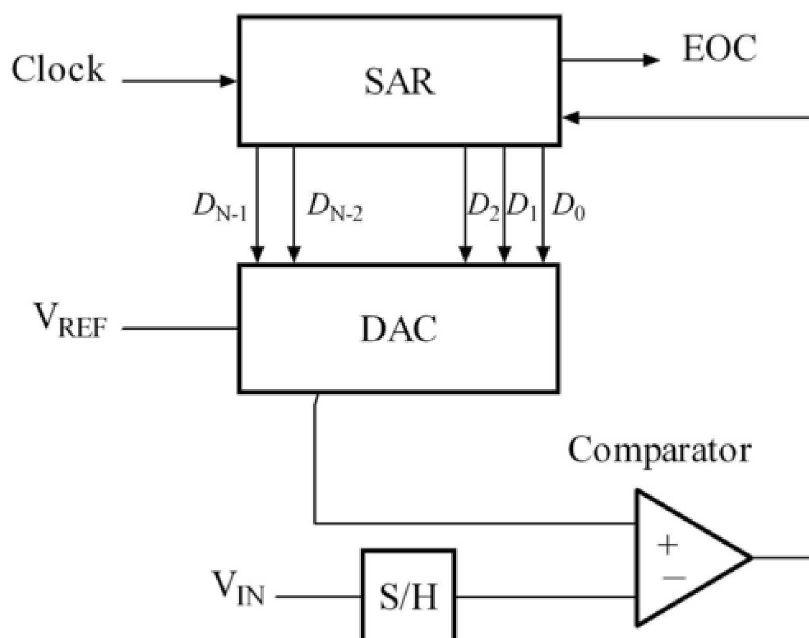
while (1); /* loop forever */
}
```

مبدل آنالوگ به دیجیتال

عمده روش هایی که برای تبدیل آنالوگ به دیجیتال وجود دارند عبارتند از: تبدیل آنـی یـا Flash، روش تقریـب هـای متـوالی یـا Successive Approximation، مبـدل هـای Delta-Encoded، Ramp-Compare، ADC Pipeline، Sigma-Delta و غیره که از این میان میکروکنترلرهای AVR از روش تقریب های متـوالی استفاده می کنند.

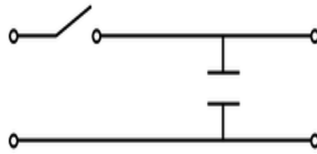
اصول تبدیل با روش تقریب های متوالی

بلوک دیاگرام ساده شده ی این مبدل به صورت زیر است:



Sample and Hold: مبدل آنالوگ به دیجیتال برای تبدیل یک نمونه ی آنالوگ به مقدار بـاینری متنـاظر بـا آن نیاز به یک ورودی Stable دارد که این طریق مدار Sample and Hold ایجاد مـی شـود. در شـکل زیـر یـک

نمونه ی بسیار ساده از آن را مشاهده می کنید، کلید، سیگنال ورودی را با هر نمونه ی برداشته شده به خازن وصل می کند و خازن نیز مقدار ولتاژ را تا نمونه ی بعدی ثابت نگاه می دارد.



Successive Approximation Register: این رجیستر مقدار تقریب زده شده ی دیجیتال را برای مقایسه به DAC می دهد.

الگوریتم تبدیل:

ابتدا رجیستر SAR با عدد باینری 10000000 بارگذاری می شود و این عـدد توسـط DAC بـا مقـدار آنـالوگ ورودی مقایسه می شود. در صورتی که عدد مقایسه شده بزرگتر باشد و خروجی مقایسه کننـده باعـث مـی شـود، SAR بیت MSB را پاک کرده و بیت قبل از آن را یک کند و در نتیجه عـدد 01000000 در DAC بارگـذاری می شود. در صورتی که عدد مقایسه شده کوچکتر باشد خروجی مقایسه کننده باعث می شـود بیـت کـوچکتر نیـز یک شده و در نتیجه عدد 11000000 در ورودی DAC بارگذاری شود. این عمل تا پیـدا شـدن مقـدار آنـالوگ ادامه داشته و در این زمان بیت End of Conversion به نشانه ی پایان تبدیل یک می شود.

برخی از مشخصات ADC قطعه ی ATmega16:

• 10-bit Resolution

• ±2 LSB Absolute Accuracy

• 65 - 260 µs Conversion Time

• Up to 15 kSPS at Maximum Resolution

• 8 Multiplexed Single Ended Input Channels

• 7 Differential Input Channels

• 2 Differential Input Channels with Optional Gain of 10x and 200x

• 0 - VCC ADC Input Voltage Range

• Selectable 2.56V ADC Reference Voltage

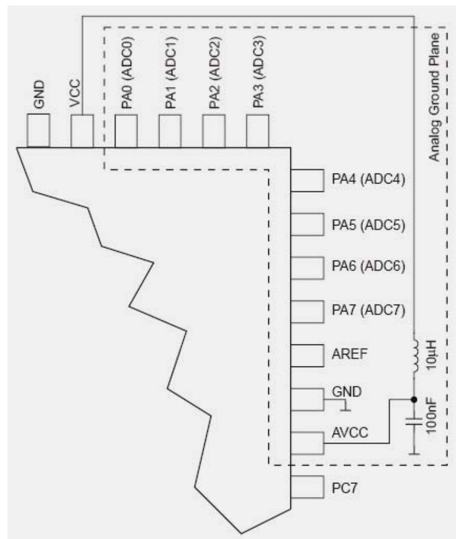
• Free Running or Single Conversion Mode

• ADC Start Conversion by Auto Triggering on Interrupt Sources

• Interrupt on ADC Conversion Complete

• Sleep Mode Noise Canceler

پین های ورودی ADC عملکرد دوم PORA می باشند که به صورت مالتی پلکس شده به ADC اعمال می شوند. ولتاژ ورودی بین صفر تا ولتاژ مرجع بوده و ولتاژ مرجع از سه منبع AVCC، پین AREF و ولتاژ داخلی ۲.۵۶ ولت قابل تامین می باشد. جهت کاهش نویز موثر بر روی واحد ADC تغذیه ی آن به صورت جداگانه از پین AVCC تامین می شود. ولتاژ این پین نباید بیشتر از ۰.۳ ولت با VCC تفاوت داشته باشد. در صورتی که از VCC به عنوان AVCC استفاده می شود، می توان بوسیله ی یک فیلتر LC این پایه به VCC متصل نمود.



رجیسترهای واحد ADC

ADC Multiplexer Selection Register

ADMUX	7	6	5	4	3	2	1	0
نام بیت	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

Analog Channel and Gain Selection Bits[4:0]: این بیت ها تعیین می کنند که چه ترکیبی از ۸ کانال ورودی به واحد ADC متصل شده و همچنین بهره ورودی تفاضلی را نیز تعیین می کنند. در حالت های Single-ended دقت ADC ۱۰ بیتی بوده که در حالت ورودی دیفرانسیل با بهره ی 1x و 10x این مقدار به ۸ بیت و با بهره ی 200x به ۷ بیت کاهش می یابد.

در صورتی که ADC مشغول انجام یک تبدیل بوده و این بیت ها تغییر کنند تا اتمام تبدیل جاری این تغییر انجام نخواهد شد.تنظیمات این ٤ بیت مطابق جدول زیر می باشد: (عملکرد تفاضلی فقط بر روی Package های TQFPو MLF آزمایش شده است.)

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010[1]		ADC0	ADC0	200x
01011[1]		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110[1]		ADC2	ADC2	200x
01111[1]		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
11101		ADC5	ADC2	1x
11110	1.22 V (V_{BG})	N/A		
11111	0 V (GND)			

ADC Left Adjust Result: بیت ADLR بر نحوه نمایش نتیجه ی تبدیل در رجیستر داده ی ADC تـاثیر می گذارد. نوشتن یک در این بیت آن را به صورت Left Adjust تنظیم می کند و در غیـر اینصـورت نتیجـه بـه صورت Right Adjust خواهد بود. تغییر این بیت به صورت آنی بر روی رجیستر داده تاثیر می گذارد.

Reference Selection Bits[1:0]: این بیت ها مرجع ولتاژ ADC را مطابق جدول زیر تعیین می کننـد. در صورتی که این بیت ها در حین تبدیل تغییر کنند تا اتمام تبدیل تغییر اعمال نخواهد شد. در صورتی کـه از مرجـع ولتاژ داخلی استفاده می شود نباید ولتاژ خارجی به پین AREF اعمال شود. زمانی که یکی از دو ولتاژ AREF یا ۲.٥٦ ولت به عنوان مرجع انتخاب شده باشند با اتصال یک خازن ۱۰۰ نانو بـین پـین AREF و زمـین مـی تـوان مقدار نویز را کاهش داد.

REFS1	REFS0	ولتاژ مرجع
۰	۰	ولتاژ پایه ی AREF
۰	۱	ولتاژ پایه ی AVCC
۱	۰	رزرو شده
۱	۱	ولتاژ داخلی ۲.٥٦ ولت

ADC Control and Status Register A

ADCSRA	7	6	5	4	3	2	1	0
نام بیت	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADC Prescaler Select Bits[2:0]: این بیت ها ضریب پیش تقسیم کننده ای را که از کلاک سیستم بـرای واحد ADC کلاک تامین می کند را مشخص می کند.

ADPS2	ADPS1	ADPS0	ضریب تقسیم
٠	٠	٠	٢
٠	٠	١	٢
٠	١	٠	٤
٠	١	١	٨
١	٠	٠	١٦
١	٠	١	٣٢
١	١	٠	٦٤
١	١	١	١٢٨

ADC Interrupt Enable: در صورت یک بودن بیت فعال ساز عمومی وقفه (I) و یک بودن این بیت، اتمام یک تبدیل می تواند باعث ایجاد وقفه شود.

ADC Interrupt Flag: با اتمام یک تبدیل این پرچم یک شده و در صورت فعال بودن وقفـه، اجـرای ISR می تواند باعث پاک شدن آن شود و در غیر اینصورت با نوشتن یک در محل این بیت می توان آن را پاک نمود.

ADC Auto Trigger Enable: عملیات تبدیل به دو صورت می تواند راه اندازی شـود، Single و Auto Trigger که حالت اول با هر بار راه اندازی ADC یک تبدیل انجام شده و در وضعیت دوم ADC بـه صـورت خودکار از طریق یکی ازمنابع داخلی تحریک می شود. برای قرار دادن ADC در وضـعیت Auto Trigger بایـد این بیت یک شود. نوع منبع تریگـر کننـده بوسـیله ی بیـت هـای ADTS[2:0] از رجیسـتر SFIOR انتخـاب می شود.

ADC Start Conversion: در وضعیت راه اندازی Single، برای آغاز هر تبدیل باید این بیت یـک شـود و در وضعیت تبدیل پیوسته (Free Running) نوشتن یک روی این بیت اولین تبدیل را موجب می شود.

ADC Enable: این بیت فعال ساز ماژول ADC بوده و با یک کردن آن می توان ADC را فعال نمود. نوشـتن صفر روی این بیت در حالی که ADC مشغول تبدیل است باعث می شود که عملیات تبدیل نیمه کاره رها شود.

The ADC Data Register

با پایان عملیات تبدیل نتیجه در این رجیستر قرار می گیرد و در صورتی که ورودی ADC به صـورت دیفرانسـیل باشد نتیجه به فرم مکمل ۲ نمایش داده می شود.

مطابق بیت ADLR در رجیستر ADMUX به دو صورت LA و RA نمایش داده می شود:

ADLR=0

Bit	7	6	5	4	3	2	1	0
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
ADCH	–	–	–	–	–	–	ADC9	ADC8

ADLR=1

Bit	7	6	5	4	3	2	1	0
ADCL	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
ADCH	ADC1	ADC0	–	–	–	–	–	–

Special FunctionIO Register

SFIOR	7	6	5	4	3	2	1	0
نام بیت	ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10

ADC Auto Trigger Source: در صورتی که بیت ADATE از رجیستر ADCSRA مقدار یک داشته باشد بیت های ADTS تعیین می کنند که کدام منبع به صورت خودکار ADC را راه اندازی کند. منبع این راه اندازی لبه ی بالا رونده ی پرچم وقفه ی آن منبع می باشد و در صورتی که بخواهیم اتمام تبدیل خود ADC منبع تریگر بعدی باشد و ADC به صورت پیوسته عملیات تبدیل را انجام دهد از حالت Free Running استفاده می کنیم.

منبع راه اندازی	ADTS0	ADTS1	ADTS2
Free Running	۰	۰	۰
مقایسه کننده ی آنالوگ	۱	۰	۰
وقفه ی خارجی صفر	۰	۱	۰
تطبیق مقایسه ی تایمر صفر	۱	۱	۰
سرریز تایمر صفر	۰	۰	۱
تطبیق مقایسه ی B تایمر یک	۱	۰	۱
سرریز تایمر یک	۰	۱	۱
Capture تایمر یک	۱	۱	۱

راه اندازی ADC به صورت تک تبدیل و تبدیل خودکار

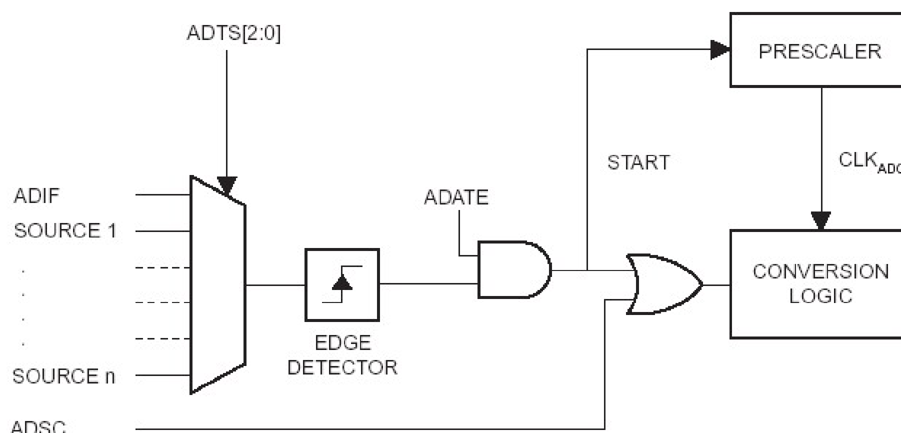
پس از انتخاب کانال مورد نظر و ولتاژ مرجع بوسیله رجیستر ADMUX، با در نظر گرفتن اینکه بیت ADEN یک بوده باشد، نوشتن یک منطقی بر روی بیت ADSC شروع یک تبدیل را موجب خواهد شد. این بیت در حین انجام تبدیل یک بوده و با پایان آن بوسیله ی سخت افزار پاک می شود. در صورتی که قبل از اتمام تبدیل، کانال ADC تغییر کند تا پایان تبدیل جاری این تغییر تاثیر نخواهد گذاشت.

راه دیگر برای راه اندازی ADC وضعیت تحریک خودکار می باشد که این حالت با یک کردن بیت ADATE از رجیستر ADCSRA آغاز می شود. منبع تریگر بوسیله ی بیت های ADTS در رجیستر SFIOR انتخاب می

شوند. زمانی که پرچم منبع تریگر یک می شود، پیش تقسیم کننده Reset شـده و ADC شـروع بـه انجـام یـک تبدیل می کند، بدین وسیله می توان در بازه های زمانی ثابت ADC را تریگر نمود.

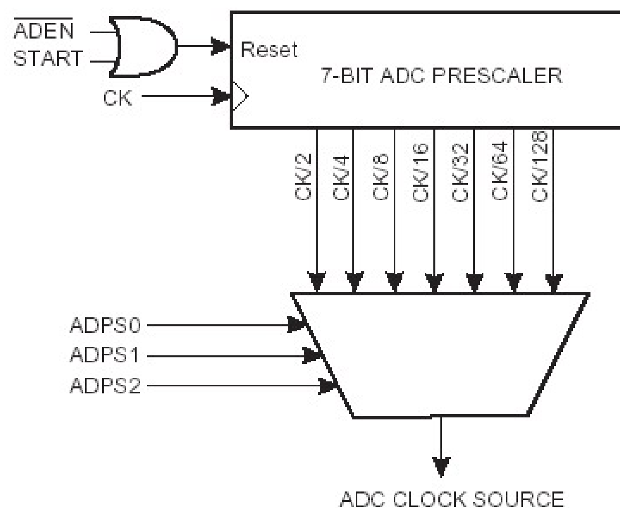
پرچم اتمام یک تبدیل بیت ADIF می باشد، در صورتی که ADC در وضعیت تحریـک خودکـار قـرار گرفتـه و این بیت به عنوان منبع تریگر انتخاب شود، پس از اتمام یک تبدیل، تبدیل جدیدی شروع خواهد شد. برای رسیدن به این وضعیت باید بیت های ADTS در حالت Free Running قرار گیرند و برای شروع تبدیل تنها یـک بـار یک کردن ADSC آغاز شده و پس از آن تبدیلات متوالی انجام خواهد شد.

همان طور که گفته شد در صورت فعال بودن بیت ADIE بالا رفتن پرچم اتمام تبدیل (ADIF) می تواند باعث ایجاد وقفه شده و با اجرای ISR این بیت توسط سـخت افـزار پـاک شـود. در صـورتی کـه ADC در وضـعیت Single Conversion یا تک تبدیل باشد برای شروع تبدیل بعدی باید پرچم ADIF با نوشتن یک بـر روی آن پاک شود، در حالیکه وضعیت Free Running انتخاب شده باشد بدون در نظر گرفتن اینکه پرچم ADIF پـاک شده است یا نه تبدیلات متوالی انجام خواهد شد.



پیش تقسیم کننده و زمان بندی تبدیل

به صورت پیش فرض، مداری که بر اساس تقریب های متوالی تبدیل آنالوگ به دیجیتال را انجام می دهد برای رسیدن به ماکزیمم Resolution، نیاز به یک کلاک ورودی با فرکانسی بین ۵۰ تا ۲۰۰ کیلوهرتز دارد. ماژول ADC برای تامین کلاک مورد نیاز دارای یک پیش تقسیم کننده می باشد که مقدار آن بوسیله ی بیت های ADPS از رجیستر ADCSRA تعیین می شود. واحد ADC برای عملکرد صحیح به فرکانس کلاکی بین ۵۰ تا ۲۰۰ کیلوهرتز نیاز دارد و در صورتی که مقدار آن خارج از این محدوده تعریف شود ممکن است عملکرد صحیحی نداشته باشد.



در صورت استفاده از Mode تک تبدیل به دلیل تنظیمات اولیه ی ADC هر تبدیل حدود ۲۵ سیکل کلاک طول می کشد، درحالیکه در وضعیت Free Running هر تبدیل برای کامل شدن حدود ۱۳ کلاک زمان نیاز دارد.

فیلتر کاهش نویز ورودی

مدارات داخلی میکروکنترلر با ایجاد نویز باعث کاهش دقت مقدار خوانده شده توسط ADC می شوند، برای بهبود این مشکل می توان در زمان تبدیل میکروکنترلر را به یکی از Mode های کم توان ADC Noise Reduction یا Idle برد تا عملیات تبدیل بعد از خاموش شدن CPU انجام شود.

پروژه ۱۳: اندازه گیری دما با سنسور LM35

```
/*****************************************************
Project : Temprature Measurement with LM35
Author  : Reza Sepas Yar
Company : Pishro Noavaran Kavosh
Chip type           : ATmega16
Clock frequency     : 1.000000 MHz
*****************************************************/

#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#define xtal 8000000

// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x12 ;PORTD
```

```
#endasm

#include <lcd.h>

#define ADC_VREF_TYPE 0xC0
// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input|ADC_VREF_TYPE;
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}

void main(void)
{

char lcd_buff[10];
int adc_in;
float temp;

PORTA=0x00;
DDRA=0x00;

// ADC initialization
// ADC Clock frequency: 45 kHz
// ADC Voltage Reference: Int., cap. on AREF
```

```
// ADC Auto Trigger Source: None
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x86;

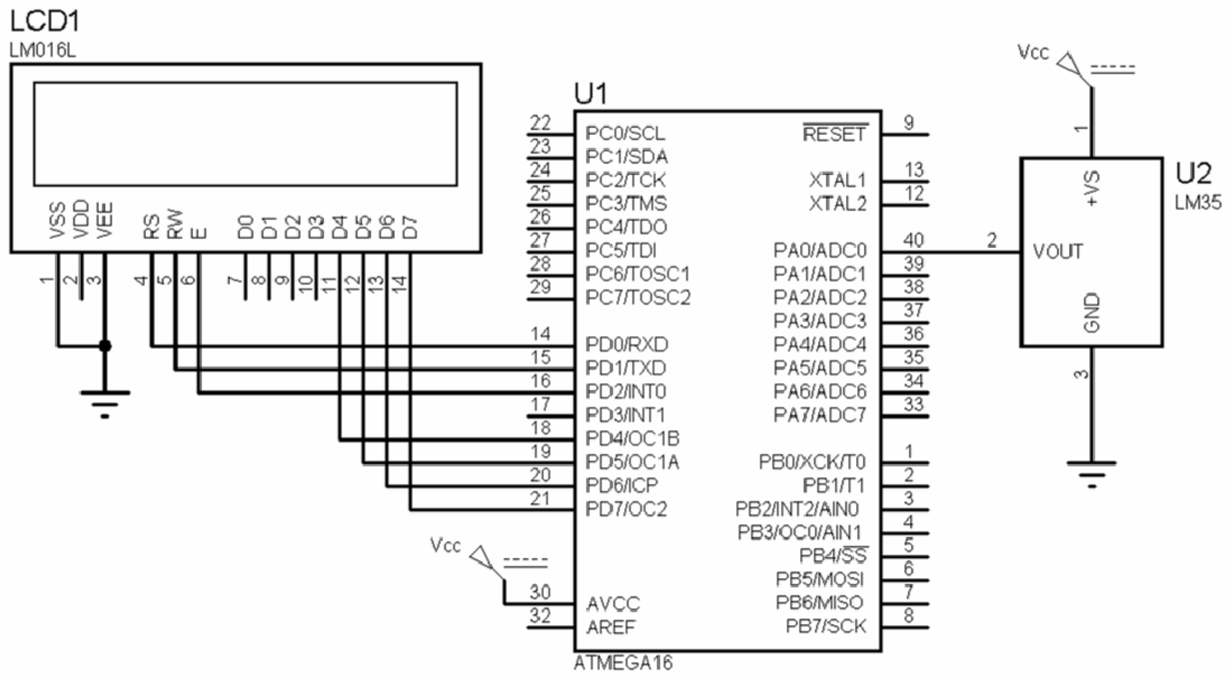
// LCD module initialization
lcd_init(16);

while (1)
      {

      adc_in=read_adc(0);
      temp=adc_in/4;
      sprintf(lcd_buff,"Temp=%5.1f C",temp);
      lcd_clear();
      lcd_gotoxy(0,0);
      lcd_puts(lcd_buff);
      delay_ms(1000);

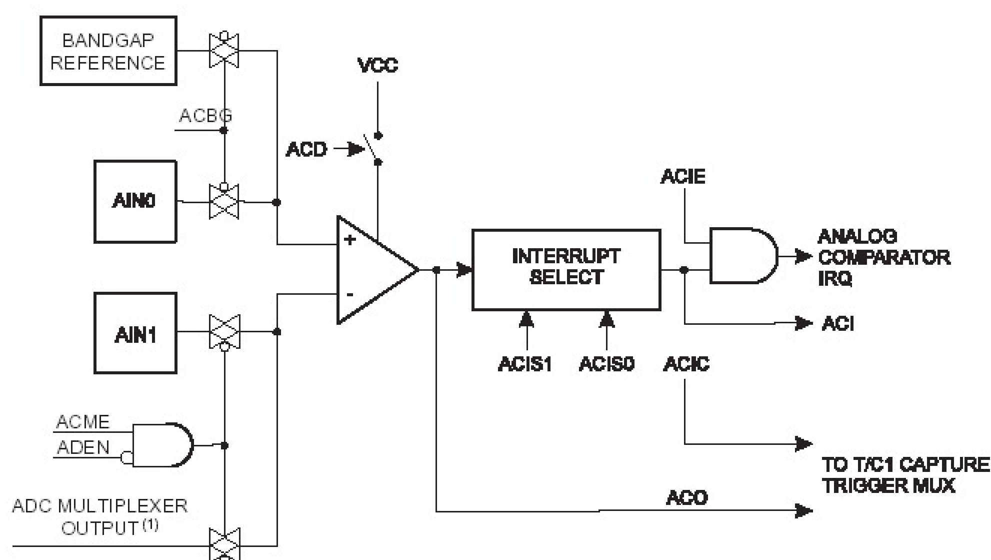
      };
}
```

شماتیک:


LCD1
LM016L
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
U1
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP
PD7/OC2
AVCC
AREF
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/XCK/T0
PB1/T1
PB2/INT2/AIN0
PB3/OC0/AIN1
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
ATMEGA16
Vcc
U2
LM35
+VS
VOUT
GND

مقایسه کننده ی آنالوگ

ماژول مقایسه کننده ی آنالوگ دارای دو ورودی AIN0 و AIN1 می باشد که این دو به ترتیب ورودی مثبـت و منفی واحد مقایسه کننده بوده که همانند یک تقویت کننده ی عملیاتی ولتاژ روی ایـن دو پایـه را مقایسـه کـرده و هنگامی که ولتاژ روی پایه ی AIN0 بیشتر از ولتاژ AIN1 باشد خروجی آن یعنی بیت ACO یک می شود. این خروجی علاوه بر کاربردهای عادی می تواند برای تریگر کردن ورودی Capture تایمر یک نیز به کار رود.



رجیسترهای مقایسه کننده ی آنالوگ

Special Function IO Register

SFIOR	7	6	5	4	3	2	1	0
نام بیت	ADTS2	ADTS1	ADTS0	-	**ACME**	PUD	PSR2	PSR10

Analog Comparator Multiplexer Enable: ماژول مقایسه کننده ی آنالوگ این امکان را می دهد تـا ورودی منفی از طریق پایه های ADC0 تاADC7 انتخاب شود. در صورت یک بودن بیت ACME و خـاموش بودن ADC (بیت ADEN در ADCSRA صفر باشد)، خروجی مالتی پلکسر ADC بـه عنـوان ورودی منفـی مقایسه کننده انتخاب می شود و در غیر اینصورت پایه AIN1 ورودی منفی خواهد بود.

Analog Comparator Control and Status Register

ACSR	7	6	5	4	3	2	1	0
نام بیت	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0

Analog Comparator Interrupt Mode Select[1:0]: این بیت ها تعیین می کننـد کـه کـدامیک از رویدادهای مقایسه کننده ی آنالوگ مطابق جدول زیر وقفه ی آن را تریگر می کند.

ACIS1	ACIS0	وضعیت وقفه
۰	۰	وقفه ی مقایسه کننده در Toggle خروجی
۰	۱	رزرو شده
۱	۰	وقفه ی مقایسه کننده در لبه ی پایین رونده ی خروجی
۱	۱	وقفه ی مقایسه کننده در لبه ی بالا رونده ی خروجی

Analog Comparator Input Capture Enable: با یک شدن این بیت ورودی Capture تایمر یک از طریق خروجی مقایسه کننده تریگر می شود.

Analog Comparator Interrupt Enable: در صورتی که این بیت و بیت فعال ساز عمومی وقفه ها یک باشد وقفه ی مقایسه کننده ی آنالوگ فعال خواهد بود.

Analog Comparator Interrupt Flag: این پرچم زمانی که یکی از رویداد های تعریف شده بوسیله ی بیت های ACIS1 و ACIS0 روی دهد، بوسیله ی سخت افزار یک می شود. در این وضعیت اگر ACIE و فعال ساز عمومی وقفه ها یک باشد برنامه به ISR مقایسه کننده منشعب خواهد شد و بیت ACI توسط سخت افزار پاک خواهد شد، در غیر اینصورت نرم افزار می تواند با نوشتن یک آن را پاک کند.

Analog Comparator Output: این بیت خروجی مقایسه کننده آنالوگ بوده و با تاخیری بین یک تا دو سیکل سنکرون می شود.

Analog Comparator Bandgap Select: با یک شدن ایـن بیـت ولتـاژ مرجـع Bandgap جـایگزین ورودی مثبت مقایسه کننده خواهد شد. در صورت صفر بودن بیـت ACGB پـین AIN0 ورودی مثبـت مقایسـه کننده خواهد بود.

Analog Comparator Disable: با یک شدن این بیت تغذیه ی مقایسه کننده ی آنالوگ قطع می شود. این مسئله به کاهش توان مصرفی در Mode های فعال و بیکاری کمک خواهد کرد. قبـل از تغییـر دادن بیـت ACD باید وقفه ی مقایسه کننده ی آنالوگ با پاک کردن بیت ACIE از ACSR غیر فعـال شـود در غیـر اینصـورت در زمان تغییر این بیت می تواند وقفه بوجود آید.

ورودی مالتی پلکس شده ی مقایسه کننده ی آنالوگ

این امکان وجود دارد که هر یک از ورودی های ADC0 تا ADC7 به عنوان ورودی منفی مقایسه کننده آنـالوگ انتخاب شوند. برای استفاده از این مسئله باید (با صفر بودن بیت ADEN) مبدل آنالوگ به دیجیتال خاموش بـوده و بیت ACME از SFIOR یک باشد. در این وضعیت بیت های MUX[2:0] از رجیستر ADMUX ورودی منفی را مطابق جدول زیر انتخاب می کنند. مسلما در صورتی که بیت ACME صفر بوده یا ADEN یـک باشـد پین AIN1 ورودی منفی مقایسه کننده ی آنالوگ خواهد بود.

ACME	ADEN	MUX[2:0]	ورودی منفی مقایسه کننده
۰	x	xxx	AIN1
x	۱	xxx	AIN1

١	٠	٠٠٠	ADC0
١	٠	٠٠١	ADC1
١	٠	٠١٠	ADC2
١	٠	٠١١	ADC3
١	٠	١٠٠	ADC4
١	٠	١٠١	ADC5
١	٠	١١٠	ADC6
١	٠	١١١	ADC7

مثال: (اعلام عبور خروجی یک سنسور آنالوگ از یک سطح معین)

```
#include <mega16.h>

// Analog Comparator interrupt service routine
interrupt [ANA_COMP] void ana_comp_isr(void)
{
PORTA=PORTA^0x01;
}

// Declare your global variables here

void main(void)
{
// Declare your local variables here
```

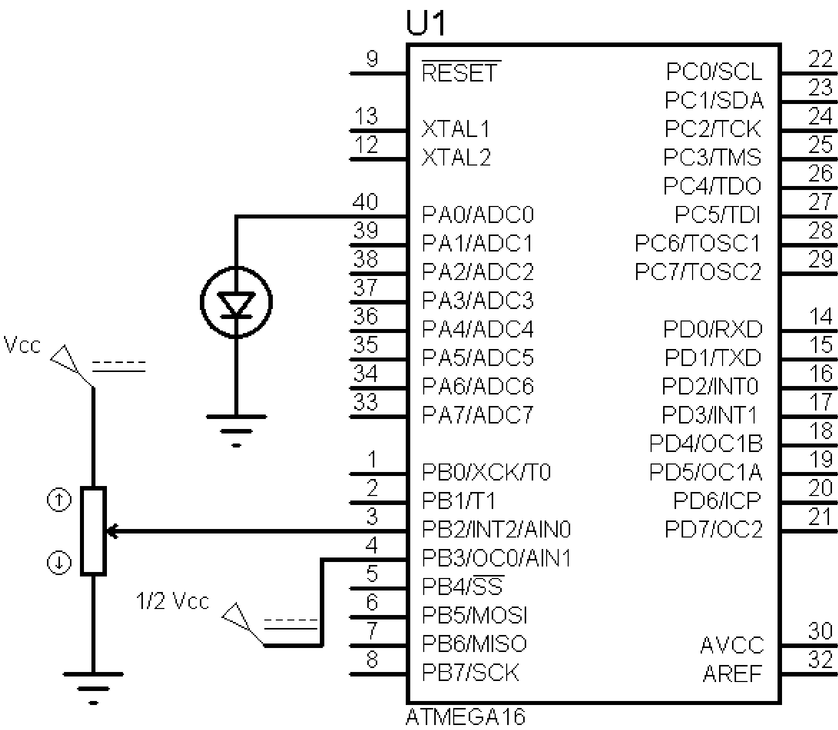
```
PORTA=0x00;
DDRA=0x01;

// Analog Comparator initialization
// Analog Comparator: On
// Interrupt on Output Toggle
// Analog Comparator Input Capture by Timer/Counter 1:
Off
ACSR=0x08;
SFIOR=0x00;

// Global enable interrupts
#asm("sei")

while (1);
}
```

شماتیک:


U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/XCK/T0
PB1/T1
PB2/INT2/AIN0
PB3/OC0/AIN1
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP
PD7/OC2
AVCC
AREF
ATMEGA16
Vcc
1/2 Vcc

SPI Bus

SPI که یک استاندارد سریال سنکرون می باشد سرنام Serial Peripheral Interface بوده و بوسیله شرکت موتورولا طراحی شده است. این استاندارد به لحاظ پشتیبانی از سرعت های بالا نه تنها در کاربردهای اندازه گیری بلکه در مواردی نظیر انتقال حجم بالای اطلاعات، پردازش سیگنال دیجیتال، کانال های ارتباطی و ... نیز مورد استفاده واقع می شود. سرعت چند مگابیت بر ثانیه به راحتی توسط SPI قابل دسترسی است و در نتیجه امکان انتقال صوت فشرده نشده و تصویر فشرده شده وجود خواهد داشت.

در زیر لیست برخی از وسایل SPI آورده شده است:

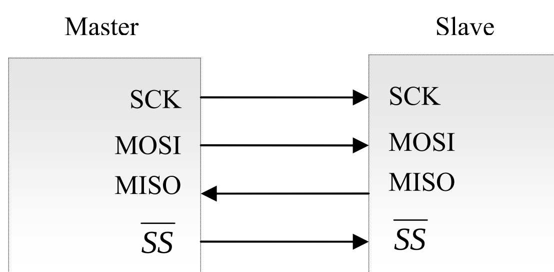
قطعه	عملکرد	مشخصات	شرکت سازنده
SSM2163	8x2 Audio Mixer	63dB attenuation in 1dB steps	Analog Devices
AD7303	DAC	8Bit clock rate up to 30MHz	Analog Devices
AD7811	ADC	10Bit 4/8 channel 300ksps	Analog Devices
AD7816	ADC + Temperature Sensor	10Bit	Analog Devices
AD7858	ADC	12Bit, 8 channel 200ksps	Analog Devices
AD8400	Digital Pot	1/2/4 channel 256 positions 1, 10, 50 100kOhm 10MHz update rate	Analog Devices
AT25010	EEPROM	Low voltage operation 1.8V/2.7V/5.0V block write protection 100 years data retention	ATMEL
AT45D011	FLASH	5V 1MBit 5MHz clock rate	ATMEL
AT45D021	FLASH	5V 2MBit 10MHz	ATMEL
AT45DB021	FLASH	2.7V 2MBit 5MHz clock rate	ATMEL

AT45DB041	FLASH	5V 4MBit 10MHz	ATMEL
AT45D081	FLASH	5V 8MBit 10MHz	ATMEL
AT45DB161	FLASH	2.7V 16MBit 13MHz	ATMEL
ADS1210	ADC	24Bit	BURR-BROWN
ADS7835	ADC	12Bit low power 500ksps	BURR-BROWN
ADS7846	Touch-screen controller	2.2V to 5.25V	BURR-BROWN
DS1267	Digital potentiometer	Dual 10k, 50k and 100k	DALLAS
DS1305	RTC	96-byte User-RAM	DALLAS
DS1306	RTC	96-byte User-RAM	DALLAS
DS1722	Digital Thermometer	-55 °C to 120 °C accuracy +/- 2°C	DALLAS
DS1844	Digital Pot	4 channel, linear 64 positions	DALLAS
NM25C020	EEPROM	data retention >40 years hard- and software write protection	Fairchild
KP100	Pressure Sensor	Range 60 ... 130kPa	Infineon
82527	CAN Controller	Flexible CPU-interface CAN 2.0	Intel
MAX349	MUX	8-to-1 dual 4-to-1	Maxim
MAX504	DAC	10Bit low power internal reference	Maxim
MAX522	DAC	8Bit 5MHz	Maxim
MAX535	DAC	13Bit Schmitt-trigger inputs	Maxim
MAX3100	UART	Up to 230kBaud Schmitt-trigger inputs	Maxim
MAX4548	Switch	Triple 3x2-crosspoint switch	Maxim
MAX4550	Switch	Dual 4x2 cross point switch	Maxim
MAX4562	Switch	Clickless Audio/Video Switch	Maxim
MAX4571	Switch	Audio/Video	Maxim
MAX4588	MUX	Dual 4 channel 180MHz bandwidth	Maxim
MAX7219	LED display driver	8-digit 10MHz clock rate digital/analog brightness control	Maxim
25AA040	EEPROM	4k max. 3MHz clock data retention >200	Microchip

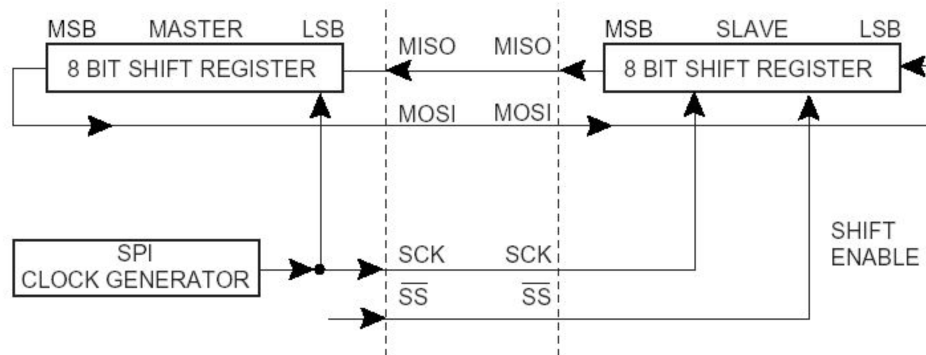
		years	
MCP3001	ADC	10Bit, 2.7V to 5V 200ksps @ 5V low power	Microchip
MCP2510	CAN Controller	Programmable Bit rate up tp 1MHz 0... 8 Bytes message frame	Microchip
MC68HC86T1	RTC + RAM	32x8Bit static-RAM	Motorola
CLC5506	GTA (Gain Trim Amplifier)	600MHz bandwidth control range 16dB	National Semiconductor
COP472-3	LCD Controller	Keine SDO-Leitung	National Semiconductor
LM74	Temperature Sensor	12Bit + sign 3V to 5V -55 °C to +150 °C	National Semiconductor
MM5483	LCD Controller	31 segment outputs cascadeable	National Semiconductor
MM58342	High Voltage Display Driver	35V max cascadeable	National Semiconductor
USBN9602	USB Controller	DMA-Support Several FIFOs	National Semiconductor

نحوه عملکرد SPI

این استاندارد برای ایجاد یک ارتباط به چهار خط ارتباطی نیاز دارد:



همانطور که ملاحظه می شود این چهار خط SCK، MOSI، MISO و $\overline{SS}$ بوده که به ترتیب خط کلاک، Master out slave in، Master in slave out و Slave Select می باشند. نحوه ی تعامل Master و Slave در شکل زیر نشان داده شده است:



سیستم شامل دو Shift Register و یک مولد کلاک می باشد. Master با صفر کردن خط $\overline{SS}$ از Slave مورد نظر، چرخه ی ارتباطی را آماده می کند. Master و Slave داده ی مورد نظر برای ارسال را در Shift Register قرار داده و Master با ایجاد کلاک در خط SCK مبادله ی داده را آغاز می کند. اطلاعات از پین MOSI در Master خارج شده و وارد پین MOSI از Slave می شود. در طرف Slave نیز داده از پین MISO خارج شده و وارد MISO از Master می شود. بعد از اتمام ارسال یک Packet، مجددا خط $\overline{SS}$ توسط Master یک شده و بدین ترتیب Slave با Master سنکرون می شود.

رجیسترهای SPI

ماژول SPI دارای سه رجیستر SPDR، SPSR و SPCR بوده که به ترتیب رجیسترهای داده، وضعیت و کنترل می باشند.

SPI Data Register

SPDR	7	6	5	4	3	2	1	0
نام بیت	MSB							LSB

نوشتن بر روی این رجیستر شروع انتقال داده را موجب خواهد شد و خواندن آن موجب خوانـدن داده ی موجـود در بافر دریافت خواهد شد.

SPI Status Register

SPSR	7	6	5	4	3	2	1	0
نام بیت	SPIF	WCOL	-	-	-	-	-	SPI2X

Double SPI Speed Bit: با نوشتن یک بر روی این بیت در صـورتیکه مـاژول SPI در وضـعیت Master باشد فرکانس کلاک موجود روی پین SCK دو برابر خواهد شد.

Write COLlision Flag: این پرچم زمانی یک خواهد شد که در حین انتقال یـک بایـت بـر روی رجیسـتر SPDR مقداری نوشته شود. با اولین خواندن رجیستر SPSR این بیت پاک می شود.

SPI Interrupt Flag: این بیت در دو حالت یک می شود: ۱. با اتمام ارسال یک بایت این پرچم یک شـده و در صورتی که بیت SPIE و فعال ساز عمومی وقفه ها یک باشند اتمام عملیات می تواند باعث ایجـاد یـک وقفـه شود. ۲. پین $\overline{SS}$ از خارج توسط یک وسیله ی دیگر زمین شود، این به معنای از دست دادن حاکمت باس بـوده و این وضعیت با یک شدن بیت SPIF اعلام می شود. با اجرای ISR یا خواندن رجیستر وضعیت این بیت پاک می شود.

SPI Control Register

SPCR	7	6	5	4	3	2	1	0
نام بیت	SPIE	SPE	DORD	MSTR	CPOL	SPHA	SPR1	SPR0

SPI Clock Rate Select 1 and 0: این دو بیت نرخ کلاک SCK را که Master ایجـاد مـی کنـد مطـابق جدول زیر تعیین می کنند.

SPI2X	SPR1	SPR0	فرکانس SCK
۰	۰	۰	f_{osc} /٤
۰	۰	١	f_{osc} /١٦
۰	١	۰	f_{osc} /٦٤
۰	١	١	f_{osc} /١٢٨
١	۰	۰	f_{osc} /٢
١	۰	١	f_{osc} /٨
١	١	۰	f_{osc} /٣٢
١	١	١	f_{osc} /٦٤

Clock Phase و **Clock Polarity**: این دو بیت مطابق جدول زیر زمـان بنـدی انتقـال و دریافـت داده روی باس SPI را تعیین می کنند:

CPOL	CPHA	لبه ی شیفت	لبه ی نمونه برداری
۰	۰	پایین رونده	بالا رونده
۰	١	بالا رونده	پایین رونده
١	۰	بالا رونده	پایین رونده
١	١	پایین رونده	بالا رونده

Master/Slave Select: با یک کردن این بیت ماژول SPI در وضعیت Master قرار می گیرد.

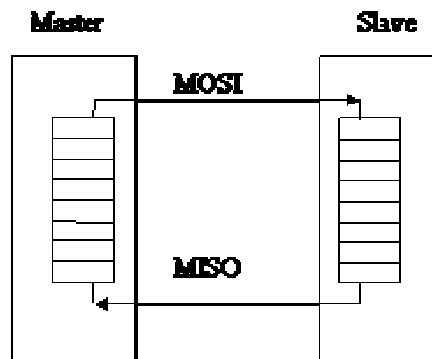
Data Order: با یک بودن این بیت ابتدا MSB روی باس منتقل می شود و در صورت صفر بودن ابتدا LSB.

SPI Enable: بیت فعال ساز SPI

SPI Interrupt Enable: بیت فعال ساز وقفه ی SPI

نحوه ی انتقال داده در SPI

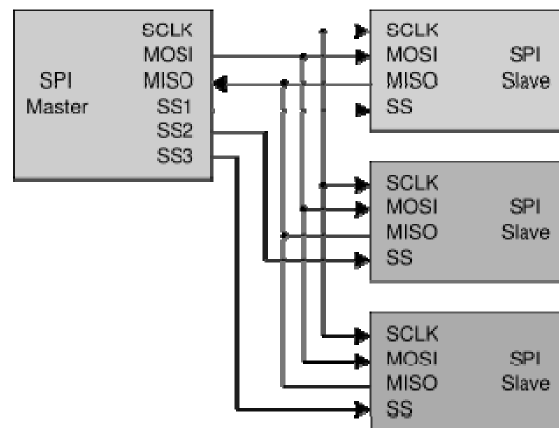
وقتی ماژول SPI به عنوان Master پیکربندی شده است خط $\overline{SS}$ را بصورت خودکار کنترل نمی کند و این وظیفه باید توسط نرم افزار، قبل از آغاز یک چرخه ی ارتباطی انجام شود. پس از صفر کردن $\overline{SS}$، نوشتن یک بایت در رجیستر داده (SPDR) باعث ایجاد کلاک توسط واحد تولید کلاک خواهد شد و با هر پالس، داده ی موجود در Shift Register های Master و Slave یک بیت شیفت داده شده و پس از ۸ پالس ساعت پرچم SPIF به نشانه ی اتمام ارسال یک می شود. پس از این Master می تواند برای ارسال بایت بعدی آن را در SPDR نوشته و یا به نشانه ی اتمام ارسال خط $\overline{SS}$ را یک نگاه دارد.



در نقطه ی مقابل زمانیکه ماژول SPI در نقش Slave پیکربندی شده است، مادامیکه خط $\overline{SS}$ یک است پین MISO در وضعیت tri-stated می باشد. در این شرایط ممکن است که رجیستر SPDR توسط Slave بروز شود اما تا زمانی که خط SS توسط Master صفر نشود انتقال انجام نخواهد شد. پس از Low شدن $\overline{SS}$ و اتمام دریافت یک بایت پرچم SPIF یک شده و در صورت یک بودن SPIE و بیت I، این رویداد می تواند باعث ایجاد وقفه شود. پس از این ممکن است Slave داده ی جدیدی را در SPDR قرار دهد منتها باید قبل از آن داده ی دریافتی را بخواند.

ارتباط شبکه ای در SPI

مطابق تصویر زیر با استفاده از پین $\overline{SS}$ می توان تعدادی Slave را کنترل نمود. Master باید تنها پین $\overline{SS}$ Slave ای را که می خواهد با آن ارتباط برقرار کند صفر کند و بقیه را یک نگه دارد.



تابع ()spi در CodeVision

اعلان این تابع در فایل spi.h بوده و نحوه ی عملکرد آن در فایل spi.lib به صورت زیر می باشد:

```
unsigned char spi(unsigned char data)
{
SPDR=data;
while ((SPSR & (1<<SPIF))==0);
return SPDR;
}
```

بنابراین آنچه این تابع به عنوان آرگومان دریافت می کند روی باس SPI قرار می دهد و مقدار بازگشتی آن مقـدار خوانده شده از باس است. قبل از استفاده از این تابع باید SPI بوسیله ی رجیسترهای کنترل و وضعیت تنظیم شده باشد.

مثال: (ارتباط دو میکروکنترلر از طریق SPI)

برنامه میکرو Master:

```
#include <mega16.h>
#include <delay.h>

// SPI functions
#include <spi.h>

void main(void)
{
```

```
unsigned char incoming;


// SCK=Out MISO=In MOSI=Out SS=Out
PORTB=0b00000000;
 DDRB=0b10110000;


// SPI initialization
// SPI Type: Master
// SPI Clock Rate: 500.000 kHz
// SPI Clock Phase: Cycle Half
// SPI Clock Polarity: Low
// SPI Data Order: MSB First
// SPI Enable: True
SPCR=0x71;


while(1)
{
        incoming=spi(0x77);
        delay_ms(50);
}
}
```

برنامه میکرو Slave:

```
#include <mega16.h>
#include <delay.h>
```

```
// SPI functions
#include <spi.h>

void main(void)
{

unsigned char incoming;

// SCK=In MISO=Out MOSI=In SS=In
DDRB=0b01000000;

// SPI initialization
// SPI Type: Slave
// SPI Clock Rate: 500.000 kHz
// SPI Clock Phase: Cycle Half
// SPI Clock Polarity: Low
// SPI Data Order: MSB First
// SPI Enable: True
SPCR=0x61;

while(1)
{
        incoming=spi(0x33);
        delay_ms(50);
}
}
```

Mode های Sleep و تایمر Watchdog

به منظور مدیریت بهینه ی توان مصرفی میکروکنترلرهای AVR دارای حداکثر ٦ Mode خواب می باشند. بـرای ورود به هریک از ٦ وضعیت خواب باید بیت SE از رجیستر MCUCE یک شده و دستورالعمل SLEEP اجرا شود. بیت های SM[2:0] از این رجیستر تعیین مـی کننـد کـه میکروکنترلـر وارد کـدامیک از Mode هـای کـم مصرف شود. در حین خواب اگر وقفه ای روی دهد میکرو از این وضعیت خارج شده و بعد از گذشـت ٤ سـیکل به علاوه ی زمان Startup، روتین سرویس وقفه را اجرا کرده و پس از آن دستورالعمل بعد از SLEEP را اجـرا خواهد کرد.

MCU Control Register

MCUCR	7	6	5	4	3	2	1	0
نام بیت	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00

Sleep Mode Select Bits[2:0]: این بیت ها مطابق جدول زیر یکی از ٦ وضعیت SLEEP را انتخاب می کنند. توجه داشته باشید که Mode های Standby و Extended Standby فقط با منبع کلاک کریستال یا رزوناتور خارجی قابل استفاده می باند.

SM2	SM1	SM0	وضعیت خواب
۰	۰	۰	Idle
۰	۰	۱	ADC Noise Reduction
۰	۱	۰	Power-down
۰	۱	۱	Power-save
۱	۰	۰	رزرو شده
۱	۰	۱	رزرو شده
۱	۱	۰	Standby
۱	۱	۱	Extende Standby

Sleep Enable: قبل از اجرای دستور SLEEP باید این بیت یک شده باشد.

Mode های خواب

- **Idle Mode:** در این وضعیت CPU متوقف شده اما SPI, USART, مقایسه کننده ی آنالوگ، ADC, TWI، تایمرها، Watchdog و سیستم وقفه به کار خود ادامه دهند. این وضعیت باعث می شود کلاک CPU و کلاک Flash متوقف شده اما بقیه ی منابع کلاک به کار خود ادامه دهند. منابع وقفه ی داخلی و خارجی می توانند باعث خروج میکروکنترلر از وضعیت خواب شوند.

- **ADC Noise Reduction Mode:** CPU متوقف شده اما ماژول های ADC, وقفه های خارجی، تشخیص آدرس TWI, تایمر ۲ و Wathdog به کار خود ادامه می دهند. در صورت فعال بودن ADC به محض ورود به این Mode انجام تبدیل شروع شده و با اتمام آن از این وضعیت خارج می شود. منابع وقفه ی ماژول هایی که در این Mode فعالند و همچنین Reset خارجی, Reset تایمر Wachdog و Brown-out Reset می توانند باعث خروج از این Mode شوند.

- **Power-down Mode:** در این وضعیت اسیلاتور خارجی متوقف شده در صورتی که وقفه های خارجی آسنکرون، TWI و Watchdog به کار خود ادامه می دهند. Reset خارجی، Reset تایمر Watchdog، Brown-out Reset، تطبیق آدرس TWI و وقفه های خارجی می توانند باعث خروج میکروکنترلر از حالت خواب شوند. اساسا در این Mode تمام کلاک ها متوقف شده و تنها ماژول های آسنکرون به کار خود ادامه می دهند.

- **Power-save Mode:** این Mode مشابه Power-down بوده با این تفاوت که اگر تایمر ۲ در Mode آسنکرون کار کند در حین خواب به کار خود ادامه خواهد داد. در صورتی که از تایمر ۲ بصورت آسنکرون استفاده نمی شود بهتر است بجای این Mode از Power-down استفاده شود.

- **Standby Mode:** این Mode مشابه Power-down بوده با این تفاوت که اسیلاتور خارجی متوقف نمی شود و اگرچه از بخش های دیگر جدا شده است اما همچنان به کار خود ادامه می دهد. در نتیجه زمان Startup حذف شده و زمان بیدار شدن میکرو به ٦ سیکل کاهش می یابد.

- **Extended Standby Mode:** این Mode مشابه Power-save بوده با این تفاوت که اسیلاتور خارجی متوقف نمی شود.

توابع مدیریت توان در CodeVision

اعلان این توابع در فایل sleep.h می باشد. شامل موارد زیر می باشند.

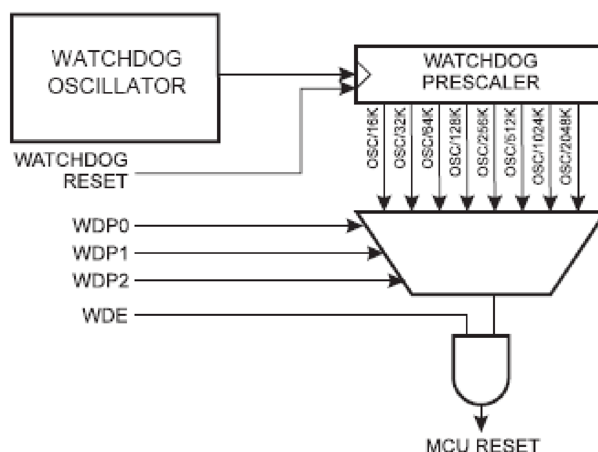
sleep_enable(): این تابع فعال ساز ورود به Mode های خواب بوده و قبل از استفاده از سایر توابع مدیریت توابع باید اجرا شود.

sleep_disable(): این تابع برای غیر فعال کردن Mode های خواب به کار می رود.

idle()، powerdown()، powersave()، standby() و extended_standby(): با اجرای هریک از این توابع میکرو وارد Mode کم توان مربوطه خواهد شد.

تایمر Watchdog

تایمر Watchdog از یک اسیلاتور داخلی مجزا با فرکانس ۱ مگاهرتز کلاک دریافت می کند که با تنظیم پیش تقسیم کننده ی تایمر Watchdog، فواصل بین هر Reset با بیت های WDP[0:2] قابل تنظیم است. با دستورالعمل WDR یا Reset شدن میکروکنترلر تایمر Watchdog, Reset شده و نرم افزار باید در فواصل مناسب با استفاده از این دستورالعمل تایمر را Reset کرده تا مانع Reset شدن میکروکنترلر شود. بلوک دیاگرام تایمر Watchdog در تصویر زیر دیده می شود.



Watchdog Timer Control Register

WDTCR	7	6	5	4	3	2	1	0
نام بیت	-	-	-	WDTOE	WDE	WDP2	WDP1	WDP0

Watchdog Timer Prescaler[2:0]: زمانی که تایمر Watchdog فعال است این بیت ها ضریب تقسیم تایمر را مطابق جدول زیر تعیین می کنند.

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 3.0V	Typical Time-out at V_{CC} = 5.0V
0	0	0	16K (16,384)	17.1 ms	16.3 ms
0	0	1	32K (32,768)	34.3 ms	32.5 ms
0	1	0	64K (65,536)	68.5 ms	65 ms
0	1	1	128K (131,072)	0.14 s	0.13 s
1	0	0	256K (262,144)	0.27 s	0.26 s
1	0	1	512K (524,288)	0.55 s	0.52 s
1	1	0	1,024K (1,048,576)	1.1 s	1.0 s
1	1	1	2,048K (2,097,152)	2.2 s	2.1 s

Watchdog Turn-off Enable و Watchdog Enable: با نوشتن یک روی بیت WDE تایمر Watchdog فعال شده و با پاک کردن آن تایمر غیر فعال می شود. اگرچه فعال کردن تایمر به سادگی و با نوشتن یک روی WDE انجام می شود اما برای غیر فعال کردن آن باید مراحل زیر به ترتیب انجام شود:

۱. همزمان بیت های WDTOE و WDE را یک کنید. (بیت WDE علیرغم اینکه قبلا یک بوده باشد باید مجددا یک شود.)

۲. تا چهار سیکل بعد نرم افزار فرصت دارد تا WDE را پاک کند.

پیوست ۱: تنظیمات رجیسترهای I/O

هر یک از چهار پورت A، B، C و D قطعه ی ATmega16 دارای سه رجیستر DDRx، PORTx و PINx بوده که x حرف مربوط به پورت می باشد. به عنوان مثال رجیسترهای اولین پورت، DDRA، PORTA و PINA بوده که وظایف هر یک در ذیل آمده است:

رجیستر Data Direction:

این رجیستر همانطور که از نام اش مشخص است رجیستر جهت داده ی پورت بوده و تعیین می کند که پورت ورودی است یا خروجی. بدین صورت که اگر روی هرکدام از بیت های این رجیستر یک نوشته شود پین متناظر آن پورت خروجی بوده و در غیر اینصورت ورودی می باشد. به عنوان مثال با اجرای عبارت DDRA = 0b10111101 وضعیت بیت های این رجیستر و پین های مربوطه به صورت زیر می باشد.

DDRA	7	6	5	4	3	2	1	0
نام بیت	1	0	1	1	1	1	0	1
جهت داده	خروجی	ورودی	خروجی	خروجی	خروجی	خروجی	ورودی	خروجی

رجیستر PORTx:

عملکرد این رجیستر بستگی به جهت داده ی پورت دارد. در صورتی که به عنوان خروجی پیکربندی شده باشد. آنچه روی پورت نوشته می شود سطح منطقی آن را تعیین می کند و در صورتی که ورودی باشد با یک کردن هر

بیت مقاومت Pull-up داخلی مربوط به آن پین فعال می شود. به عنوان نمونه در ادامه ی مثال قبل در صورتی که عبارت PORTA = 0b11010100 اجرا شود، وضعیت پورت به صورت زیر خواهد بود.

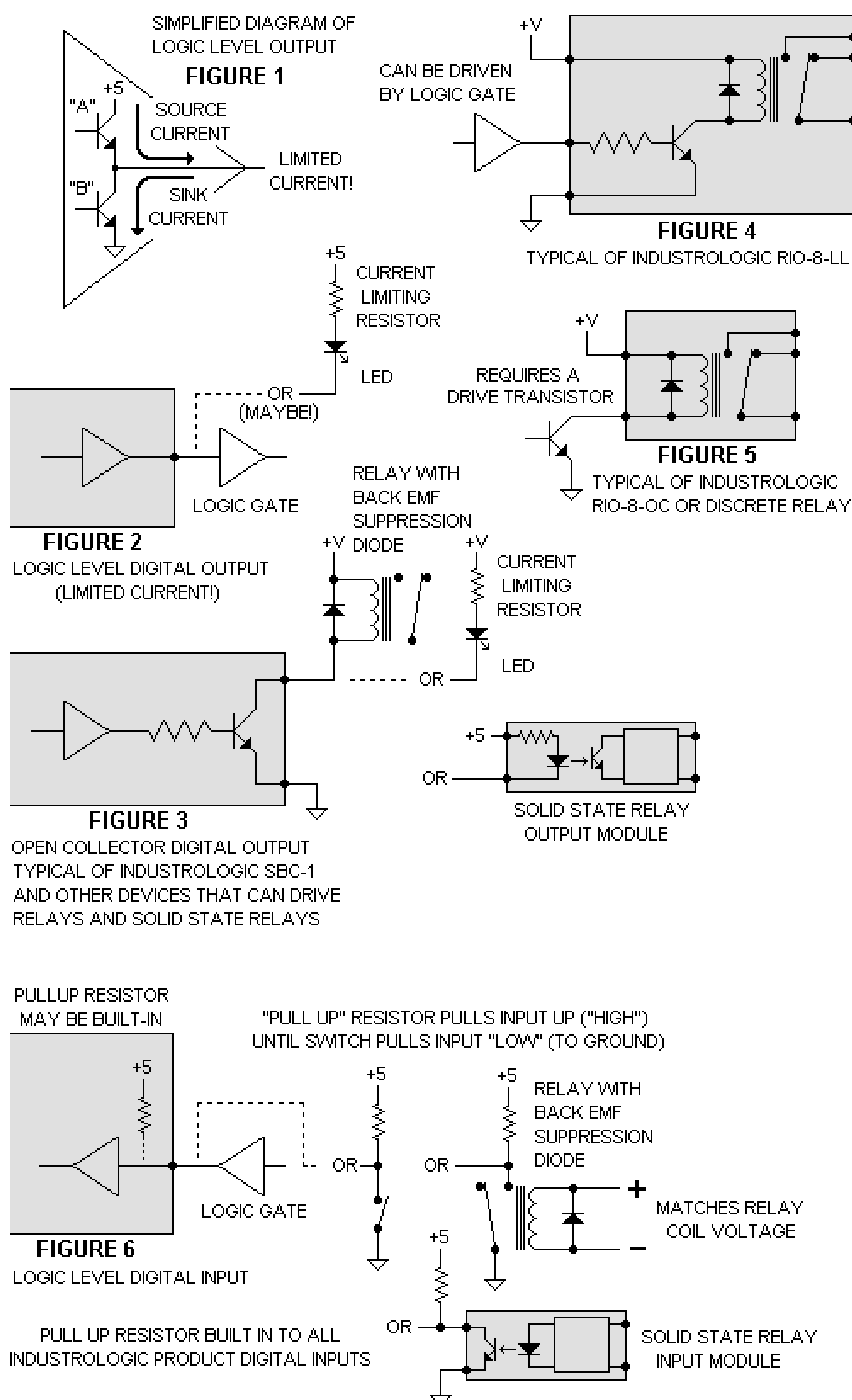
شماره بیت	7	6	5	4	3	2	1	0
DDRA	1	0	1	1	1	1	0	1
PORTA	1	1	0	1	0	1	0	0
جهت داده	خروجی با سطح منطقی یک	ورودی با مقاومت Pull-up	خروجی با سطح منطقی صفر	خروجی با سطح منطقی یک	خروجی با سطح منطقی صفر	خروجی با سطح منطقی یک	ورودی بدون مقاومت Pull-up	خروجی با سطح منطقی صفر

رجیستر PINx:

برای خواندن مقدار هر پین باید محتویات این رجیستر خوانده شود. به عنوان مثال چنانچه PORC را قبلا به صورت ورودی پیکربندی کرده باشیم و مقدار رجیستر PINC برابر 0b11010000 باشد، سطح منطقی اعمال شده به پین به صورت زیر می باشد:

PINC	7	6	5	4	3	2	1	0
نام بیت	1	1	0	1	0	0	0	0
جهت داده	یک	یک	صفر	یک	صفر	صفر	صفر	صفر

پیوست ۲: نحوه ی ارتباط دهی ورودی وخروجی های میکروکنترلر



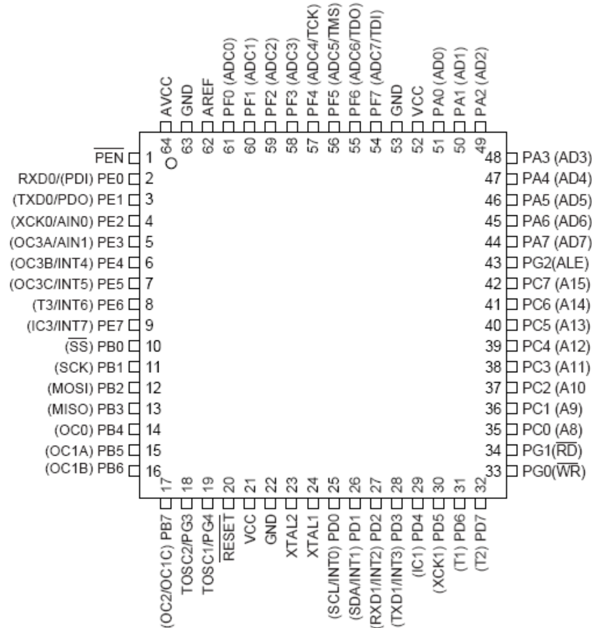
پیوست ۳: مشخصات برخی قطعات AVR

Devices	Flash (Kbytes)	EEPROM (Kbytes)	SRAM (Bytes)	Max I/O	F.max (MHz)	Vcc (V)	16-bit Timers	8-bit Timer	PWM (channels)	SPI	UART	TWI	10-bit A/D	Interrupts	Ext Interrupts
AT90PWM1	8	0.5	512	19	16	2.7-5.5	1	1	7	1	No	--	8	--	4
AT90PWM2	8	0.5	512	19	16	2.7-5.5	1	1	7	1	Yes	--	8	--	4
AT90PWM3	8	0.5	512	27	16	2.7-5.5	1	1	10	1	Yes	--	11	--	4
ATmega128	128	4	4096	53	16	2.7-5.5	2	2	8	1	2	Yes	8	34	8
ATmega1280	128	4	8192	86	16	1.8-5.5	4	2	16	1	4	Yes	16	57	32
ATmega1281	128	4	8192	54	16	1.8-5.5	4	2	9	1	2	Yes	8	48	17
ATmega16	16	0.5	1024	32	16	2.7-5.5	1	2	4	1	1	Yes	8	20	3
ATmega162	16	0.5	1024	35	16	1.8-5.5	2	2	6	1	2	--	--	28	3
ATmega164P	16	0.512	1024	32	20	1.8-5.5	1	2	6	1	2	Yes	8	31	32
ATmega165	16	0.5	1024	54	16	1.8-5.5	1	2	4	1	1	USI	8	23	17
ATmega165P	16	0.5	1024	54	16	1.8-5.5	1	2	4	1	1	USI	8	23	17
ATmega168	16	0.5	1024	23	20	1.8-5.5	1	2	6	1	1	Yes	8	26	26
ATmega169	16	0.5	1024	54	16	1.8-5.5	1	2	4	1	1	USI	8	23	17
ATmega169P	16	0.5	1024	54	16	1.8-5.5	1	2	4	1	1	USI	8	23	17
ATmega2560	256	4	8192	86	16	1.8-5.5	4	2	16	1	4	Yes	16	57	32
ATmega2561	256	4	8192	54	16	1.8-5.5	4	2	9	1	2	Yes	8	48	17
ATmega32	32	1	2048	32	16	2.7-5.5	1	2	4	1	1	Yes	8	19	3
ATmega324P	32	1	2048	32	20	1.8-5.5	1	2	6	1	2	Yes	8	31	32
ATmega325	32	1	2048	54	16	1.8-5.5	1	2	4	1	1	USI	8	23	17
ATmega3250	32	1	2048	69	16	1.8-5.5	1	2	4	1	1	USI	8	32	17
ATmega3250P	32	1	2048	69	20	1.8-5.5	1	2	4	1	1	USI	8	32	17
ATmega325P	32	1	2048	54	20	1.8-5.5	1	2	4	1	1	USI	8	23	17
ATmega329	32	1	2048	54	16	1.8-5.5	1	2	4	1	1	USI	8	25	17
ATmega3290	32	1	2048	69	16	1.8-5.5	1	2	4	1	1	USI	8	25	32
ATmega3290P	32	1	2048	69	20	1.8-5.5	1	2	4	1	1	USI	8	25	32
ATmega329P	32	1	2048	54	20	1.8-5.5	1	2	4	1	1	USI	8	25	17
ATmega406	40	0.512	2048	18	1	4-25	1	1	1	--	--	Yes	--	23	4
ATmega48	4	0.256	512	23	20	1.8-5.5	1	2	6	1	1	Yes	8	26	26
ATmega64	64	2	4096	54	16	2.7-5.5	2	2	8	1	2	Yes	8	34	8
ATmega640	64	4	8192	86	16	1.8-5.5	4	2	16	1	4	Yes	16	57	32
ATmega644	64	2	4096	32	20	1.8-5.5	1	2	6	1	1	Yes	8	31	32
ATmega644P	64	2	4096	32	20	1.8-5.5	1	2	6	1	2	Yes	8	31	32
ATmega645	64	2	4096	54	16	1.8-5.5	1	2	4	1	1	USI	8	23	17

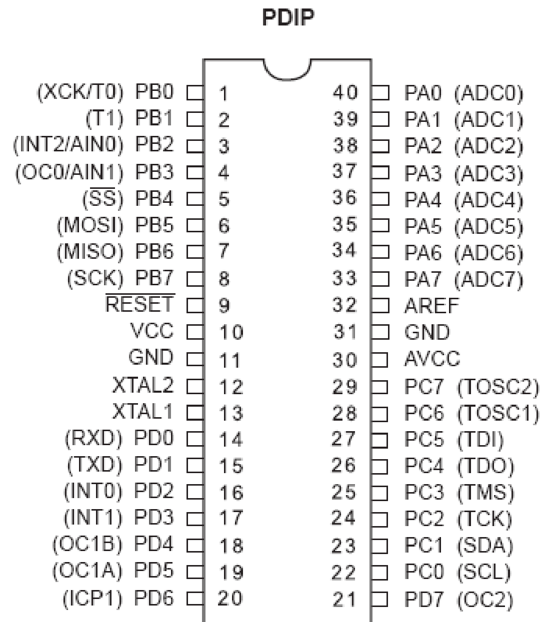
ATmega6450	64	2	4096	69	16	1.8-5.5	1	2	4	1	1	USI	8	32	17
ATmega649	64	2	4096	54	16	1.8-5.5	1	2	4	1	1	USI	8	25	17
ATmega6490	64	2	4096	69	16	1.8-5.5	1	2	4	1	1	USI	8	25	32
ATmega8	8	0.5	1024	23	16	2.7-5.5	1	2	3	1	1	Yes	8	18	2
ATmega8515	8	0.5	512	35	16	2.7-5.5	1	1	3	1	1	--	--	16	3
ATmega8535	8	0.5	512	32	16	2.7-5.5	1	2	4	1	1	Yes	8	20	3
ATmega88	8	0.5	1024	23	20	1.8-5.5	1	2	6	1	1	Yes	8	26	26
ATtiny11	1	--	--	6	6	2.7-5.5	--	1	--	--	--	--	--	4	1
ATtiny12	1	0.064	--	6	8	1.8-5.5	--	1	--	--	--	--	--	5	1
ATtiny13	1	0.064	64B	6	20	1.8-5.5	--	1	2	--	--	--	4	9	6
ATtiny15L	1	0.0625	--	6	1.6	2.7-5.5	--	2	1	--	--	--	4	8	1(+5)
ATtiny2313	2	0.128	128	18	20	1.8-5.5	1	1	4	USI	1	USI	--	8	2
ATtiny24	2	0.128	128	12	20	1.8-5.5	1	1	4	USI	--	USI	8	17	12
ATtiny25	2	0.128	128	6	20	1.8-5.5	--	2	4	USI	--	USI	4	15	7
ATtiny26	2	0.125	128	16	16	2.7-5.5	--	2	2	USI	--	USI	11	11	1
ATtiny261	2	0.128	128	16	20	1.8-5.5	1	2	2	Yes	--	USI	11	19	2
ATtiny28L	2	--	32	11	4	1.8-5.5	--	1	--	--	--	--	--	5	2(+8)
ATtiny44	4	0.256	256	12	20	1.8-5.5	1	1	4	USI	--	USI	8	17	12
ATtiny45	4	0.256	256	6	20	1.8-5.5	--	2	4	USI	--	USI	4	15	7
ATtiny461	4	0.256	256	16	20	1.8-5.5	1	2	2	Yes	--	USI	11	19	2
ATtiny84	8	0.512	512	12	20	1.8-5.5	1	1	4	USI	--	USI	8	17	12
ATtiny85	8	0.512	512	6	20	1.8-5.5	--	2	4	USI	--	USI	4	15	7
ATtiny861	8	0.512	512	16	20	1.8-5.5	1	2	2	Yes	--	USI	11	19	2

پیوست ٤: Pinout برخی قطعات AVR

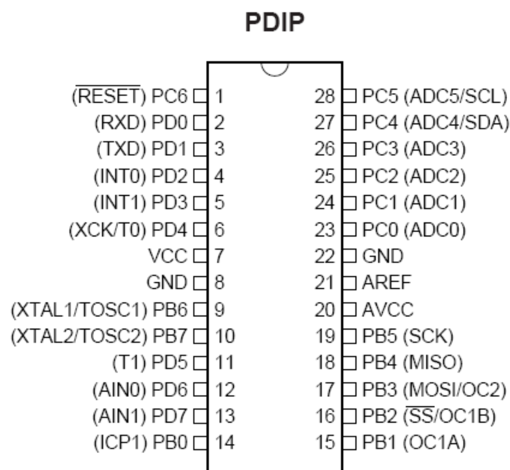
ATmega64,ATmega128



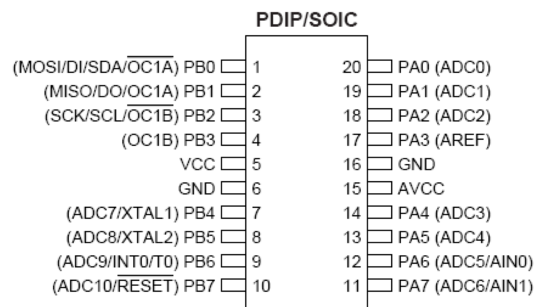
ATmega16, ATmega32



ATmega8



ATiny26



پیوست ۵: خلاصه ی رجیسترهای ATmega16

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	7
$3E ($5E)	SPH	–	–	–	–	–	SP10	SP9	SP8	10
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	10
$3C ($5C)	OCR0	Timer/Counter0 Output Compare Register								83
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	46, 67
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–	68
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	83, 114, 132
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	84, 115, 132
$37 ($57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	249
$36 ($56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	178
$35 ($55)	MCUCR	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	30, 66
$34 ($54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	39, 67, 229
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	81
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)								83
$31[1] ($51)[1]	OSCCAL	Oscillator Calibration Register								28
	OCDR	On-Chip Debug Register								225
$30 ($50)	SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	55,86,133,199,219
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	109
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	112
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte								113
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								113
$2B ($4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte								113
$2A ($4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte								113
$29 ($49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte								113
$28 ($48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte								113
$27 ($47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte								114
$26 ($46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte								114
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	127
$24 ($44)	TCNT2	Timer/Counter2 (8 Bits)								129
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register								129
$22 ($42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	130
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	41
$20[2] ($40)[2]	UBRRH	URSEL	–	–	–	UBRR[11:8]				165
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	164
$1F ($3F)	EEARH	–	–	–	–	–	–	–	EEAR8	17
$1E ($3E)	EEARL	EEPROM Address Register Low Byte								17
$1D ($3D)	EEDR	EEPROM Data Register								17
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	17
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	64
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	64
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	64
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	64
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	64
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	64
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	65
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	65
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	65
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	65
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	65
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	65
$0F ($2F)	SPDR	SPI Data Register								140
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	140
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	138
$0C ($2C)	UDR	USART I/O Data Register								161
$0B ($2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	162
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	163
$09 ($29)	UBRRL	USART Baud Rate Register Low Byte								165
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	200
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	215
$06 ($26)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	217
$05 ($25)	ADCH	ADC Data Register High Byte								218
$04 ($24)	ADCL	ADC Data Register Low Byte								218
$03 ($23)	TWDR	Two-wire Serial Interface Data Register								180
$02 ($22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	180
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	179
$00 ($20)	TWBR	Two-wire Serial Interface Bit Rate Register								178